Quantum-Inspired Hyper-Heuristics for Energy-Aware Scheduling on Heterogeneous Computing Systems

Shaomiao Chen, Zhiyong Li, *Member, IEEE*, Bo Yang, and Günter Rudolph, *Member, IEEE*

Abstract—Power and performance tradeoff optimization is one of the most significant issues on heterogeneous multiprocessor or multicomputer systems (HMCSs) with dynamically variable voltage. In this paper, the problem is defined as energy-constrained performance optimization and performance-constrained energy optimization. Task scheduling for precedence-constrained parallel applications represented by a directed acyclic graph (DAG) in HMCSs is an NP-HARD problem. Over the last three decades, several task scheduling techniques have been developed for energy-aware scheduling. However, it is impossible for a single task scheduling technique to outperform all other techniques for all types of applications and situations. Motivated by these observations, hyper-heuristic framework is introduced. Moreover, a quantum-inspired high-level learning strategy is proposed to improve the performance of this framework. Meanwhile, a fast solution evaluation technique is designed to reduce the computational burden for each iteration step. Experimental results show that the fast solution evaluation technique can improve average algorithm search speed by 38 percent and that the proposed algorithm generally exhibits outstanding convergence performance.

Index Terms—Power and performance tradeoff optimization, precedence-constrained parallel application, energy-aware scheduling, heterogeneous multiprocessor or multicomputer systems, hyper-heuristics, quantum computing

1 Introduction

High-performance clusters have served as primary and cost-effective infrastructures for complicated scientific and commercial applications, and the number of processors often maybe as many as hundreds of thousands with the system scale development. The problem of high energy consumption in high-performance clusters, which causes high costs, system instability and environmental pollution, has become more and more serious. Given the significance of the problem, various high-level approaches for reducing power consumption in multiprocessor or multicomputer systems (MCSs) have been investigated, such as energy-aware scheduling and resource allocation, dynamic server provisioning, consolidation and virtualization. Energy-aware scheduling and resource allocation is considered to be one of most promising approaches for saving energy in MCSs (e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9]). Specifically, the dynamic voltage scaling (DVS) techniques, which has been proven to be a practical technique [10], [11], has been used extensively in energy-aware scheduling (e.g., [4], [5], [6], [7], [12], [13]). Although so many algorithms and strategies have been developed for power reduction on MCSs, challenges have been detected that require further in-depth study.

(1) *Flexibly management of energy consumption in heterogeneous multiprocessor or multicomputer systems (HMCSs)*. Some types of MCSs, such as cloud, are commercial computing and service models, and often consist of heterogeneous resources that meet a need for a requested service or may be generated as a result of computation resource upgrades. Energy consumption in these types of MCSs is typically the most significant contributing factor of the operational expenses [14], [15]. Thus, flexible energy consumption management in HMCSs benefits both users and service providers. Nonetheless, most of energy-aware algorithms have been developed to improving energy efficiency ratio [1], [5], [6], and not for managing energy consumption flexibly. Although some studies have been conducted on the flexible management of energy consumption, the scopes of these works are restricted on homogeneous computing systems [2], [4], independent task scheduling [16], [17], or need too high computation cost [7], [18].

(2)*Voltage setting for a given tasks in task scheduling*. The task scheduling of applications represented by a directed acyclic graph (DAG) on MCSs is an NP-hard problem [19] and the number of choices for voltage settings on such systems grows exponentially [20]. The task scheduling on DVS-enabled MCSs is generally divided into several phases [4], [5], [6], namely, task-prioritizing, processor-selection, and voltage-dispatching phases, without considering the interaction effect among phases. However, effective task priority, processor selection, and reasonable voltage dispatching may not always generate good schedules. These problems have plagued traditional methods considerably,

- *S. Chen, Z. Li, and B. Yang are with the College of Computer Science and Electronic Engineering of Hunan University, National Supercomputing Center in Changsha, Changsha 410082, China.*
  E-mail: {chensm1987, zhiyong.li, bo_yang}@hnu.edu.cn.
- *G. Rudolph is with the Department of Computer Science, TU Dortmund University, Dortmund 44227, Germany.*
  E-mail: Guenter.Rudolph@tu-dortmund.de.

Manuscript received 30 Oct. 2014; revised 6 July 2015; accepted 9 July 2015. Date of publication 29 July 2015; date of current version 18 May 2016.
Recommended for acceptance by R. Cumplido.
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.2015.2462835

such as list scheduling algorithm [1], [6], [21] and duplication-based algorithm [5], [22], [23].

The constructive heuristic scheduling method, which is based on greedy local optimal selection by some heuristic strategies, is not very agile for different application situations. These approaches may get better performance in one type of task situation but worsen performance in another. The use of only basic constructive heuristics in complex scheduling problems often produces unacceptable solutions. Nevertheless, meta-heuristic algorithms, such as (GA) [24], [25], artificial immunity [26], simulated annealing [27], and ant colony algorithm [28], are increasingly popular in DAG task scheduling problems because of high adaptability of these algorithms. In addition, these algorithms usually generate output schedules of better quality than those generated by constructive heuristic-based ones [3], [21], [22]. However, the computation cost of these algorithms is higher than that of constructive heuristic algorithms due to the poor search efficiency and frequent solutions evaluation. According to “No Free Lunch” theorem [29], a single search methodology to achieve better performance for all kinds of applications and situations is impossible. However, this theorem does not postulate that a search methodology that can improve performance in special applications or situations cannot be developed. Automatically selecting an appropriate scheduling strategy for different situations may be a good choice to improve the search efficiency of scheduling algorithms. Moreover, the time complexity must be reduced in the search and evaluation of solution spaces for these approaches on the basis of guided search techniques.

In our paper, power-performance tradeoff problem in DVS-enabled HMCSs is defined as energy-constrained performance optimization ($Pro1$) and performance-constrained energy optimization ($Pro2$). In order to tackle the two problems, a algorithm, named quantum-inspired hyper-heuristics algorithm (QHA), is proposed. This algorithm does not employ complex search strategy to search the solution space directly; rather, it introduces the hyper-heuristic framework [30] to manage reduced low-level heuristics. In our approach, low-level heuristics are effectively designed for various specific situations or objectives. Moreover, a quantum-inspired high-level strategy that improves on the quantum-inspired evolution algorithm [31] is developed to automatically manage such low-level heuristics. Furthermore, a fast solution evaluation technique is established to reduce the computation cost of the algorithm. The main contributions of this study are as follows:

- *Problem modeling*. The power-performance tradeoff problem in DVS enabled HMCSs for precedence-constrained parallel applications scheduling is defined as energy-constrained performance optimization ($Pro1$) and performance-constrained energy optimization ($Pro2$) to manage the energy consumption flexibly.
- Hyper-heuristic framework is introduced and a reduced low-level heuristics set established. To the best of our knowledge, this framework is the first one introduced to scheduling problem in HMCSs.
- *Quantum-inspired hyper-heuristics Algorithm*. To improve the performance of hyper-heuristic framework, a quantum-inspired high-Level learning strategy is designed to evaluate the current performance of the low-level heuristics more accurately than the traditional high-level heuristic strategy can do. This strategy can also maintain the diversity of low-level selection probability.
- *Fast solution evaluation technique*. The technique can avoid double counting while evaluating solutions. The experimental result shows that the proposed method improved the search speed by 38 percent on average to traditional solution evaluation method.

The rest of this paper is organized as follows: in the following section, the works related to our research is introduced. In Section 3, the mathematical models of the problem are proposed. In Section 4, our methodology is described in detail. In Section 5, we demonstrate the workability of QHA through evaluations of simulation experiments. In Section 6, the conclusion of this study is drawn.

2 Related Work

2.1 Task Scheduling and Energy-Aware Scheduling

Static scheduling on MCSs has been researched extensively. The proposed algorithms can be classified into: list-based, clustering based, duplication-based, and guided random search-based. List-based algorithm involves the task priority list building and tasks mapping [21], [32]. The performance of these algorithms usually varies according to the different rules of each task priority list and task mapped. Cluster-based and duplication-based scheduling algorithms typically reduce communication overheads among intercommunication tasks, which prefer to communication-intensive application scheduling [33], [34]; Guided random search-based scheduling algorithms are popular in DAG scheduling [25], [26], [27], [28]. These algorithms use guided search strategy that is inspired by biological evolution or biobehavioral to search the problem space directly. These algorithms usually generate output schedules of better quality than those produced by other algorithm categories. However, the scheduling time of guided random search-based scheduling algorithms is also higher than those of others because of poor search efficiency and solution evaluation.

Energy-aware scheduling and resource allocation is one of most promising approaches to saving energy in MCSs, and DVS techniques has been widely used in energy-aware scheduling (e.g., [1], [2], [3], [4], [5], [6], [7], [8] [12], [13], [35], [36]). The proposed algorithms can be classified as follows according to different optimization objective models: energy efficiency ratio optimization algorithm, energy-constrained performance or performance-constrained energy optimization algorithm, multi-objective optimization algorithm. In an energy efficiency ratio optimization algorithm (e.g., [5], [6], [8]), a schedule is first generated, and then scrutinize the schedule, if changes to the schedule can reduce energy consumption, a new schedule is produced. Thus, the algorithm can improve energy efficiency ratio significantly, but it cannot manage energy consumption flexibly; in energy-constrained performance optimization or performance-constrained energy optimization algorithm, the target system and the task models of most algorithm do

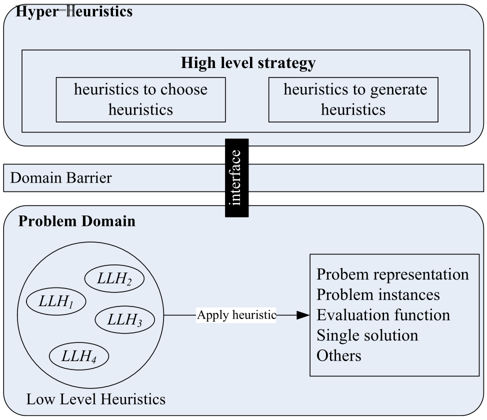


Fig. 1. Hyper-heuristic framework.

not consider heterogeneous systems and dependent tasks simultaneously. For instance, the target system model is composed of homogeneous multiprocessor systems in [2], [4]. In [16], [17] the target task model is composed of independent tasks; the multi-objective optimization-based algorithm is another way to manage energy consumption flexibly [7], [18]. However, the Pareto optimal solution set must be determined for the algorithm, and the computation cost of this process is high because of the non-dominated solution sorting operation. From the above knowledge, flexibly managing of energy consumption in DVS-enabled heterogeneous multiprocessor or multi-computer systems is challenging for scheduling dependent tasks.

2.2 Hyper-Heuristic

Different heuristics have varying capabilities that may change over the search space. In other words, no heuristic can perform ideally in all situations [29]. Hyper-heuristics are intended to provide a method to automatically managing low-level heuristics sets, as opposed to searching the solution space directly. This framework increases the generalizability of an algorithm with higher search efficiency for a range of problems [30], [37]. A generic hyper-heuristic framework is shown in Fig. 1. This framework is composed of high-levels heuristics and problem domain, which are separated by a domain barrier. Domain-independent information acquisition and processing are performed through interface transfer. The high-level strategy could be classified into two main types, “heuristics to choose heuristics” and “heuristics to generate heuristics.”

In the first category, the hyper-heuristic framework is provided with a set of preexisting heuristics. The main challenge encountered in this category of hyper-heuristics is the automatic selection of an appropriate heuristic from low-level heuristic set to search solution space. Many approaches have been proposed to address this problem. For instance, Burke et al. [38] proposed a tabu search-based hyper-heuristic, which uses a tabu list of heuristics to prevent the selection of certain heuristics at certain times during the search process. In [39], a case-based heuristic selection approach for timetabling problem was proposed. Simulated annealing is used as a hyper-heuristic in [40] for the shipper rationalization problem. Burke et al. [41] proposed Monte Carlo-based hyper-heuristics for timetabling problem examination. Gascón-Moreno et al. [42] proposed an evolutionary-based hyper-heuristic approach to optimize the construction of a group method of data handling networks.

TABLE 1
Notation Used in Mathematical Models

Notation	Definition
t_i	ith task of an application
c_i	ith processor in a system
e_{ij}	The directed edge from t_i and t_j
CC_{ij}	The communication cost between task t_i and t_j
B_{ku}	The communication rate between c_k and c_u
L_k	The communication startup costs of c_k
$data_{ij}$	The inter-task communication length between t_i and t_j
V_{kr}	The voltage corresponding to the rth voltage supply level of processor c_k
W_{ikr}	The computation cost of task t_i on processor c_k given supply voltage V_{kr}
$succ(t_i)$	The set of immediate successors of task t_i
$pred(t_i)$	The set of immediate predecessors of task t_i
$EFT(t_i)$	The earliest execution finish time of task t_i
$EST(t_i)$	The earliest execution start time of task t_i
U_k	The scheduled task set on c_k
$g(k)$	The number of VSLs that processor c_k supplies
t_{entry}	The entry task in DAG graph
t_{exit}	The exit task in DAG graph

The second category of hyper-heuristics is used to construct a high-quality heuristic for solving a target problem. For example, Burke et al. [43] employed genetic programming to evolve bin-packing heuristics. Fukunaga [44] utilized a hyper-heuristic to discover the local search heuristics for the problem of satisfiability testing. Hyde et al. [45] applied reusable heuristic genetic programming to the 2-D strip packing problem.

To the best of our knowledge, hyper-heuristics have not been used thus far to address energy-aware scheduling problems on HMCSs.

3 Mathematical Models

This section discusses the mathematical model of the precedence-constrained parallel applications energy-aware scheduling problem on HMCSs, in which processors supply with different voltage supply levels (VSLs). We summarize the notation of mathematical models in Table 1.

3.1 Application Model

An application in HMCSs is represented by a DAG, which has been used extensively to represent the precedence-constrained parallel application in programming model, such as MapReduce, Message Passing Interface (MPI). A DAG with n tasks can be defined as $G=< T,E >$, where $T=\{t_1,t_2,\ldots,t_n\}$is the set of tasks. $E=\{e_{ij}|\ 1\leq i\leq n,1\leq j\leq n\ \}$ is set of edges among tasks, where $e_{ij}\in 0,1$, and $e_{ij}=1$ represents the t_i is immediate predecessor of t_j which should complete its execution before t_j. Moreover, $data_{ij}$ represents the inter-task communication length between t_i and t_j. t_{entry} is a task without predecessors and t_{exit} is a task without successors. We assume that DAG has exactly one entry task t_{entry} and exactly one exit task t_{exit}.

3.2 Computing System Model

A HMCSs with fully interconnected m processors or machines is defined as $S = <C, B, L>$. $C = \{c_1, c_2, \ldots, c_m\}$ is the set of DVS-enabled heterogeneous computational processors. Processor c_k has $g(k)$ different types of voltage supply levels. Let $V_k = (V_{k1}, \ldots, V_{kg(k)})$ be the voltage supply vector of c_k, where V_{kr} is voltage corresponding to the rth voltage supply level of processor c_k. When processor c_k is idle, the voltage supplied is minimal (i.e.,$V_{kg(k)}$). $B$is communication rate matrix among computational nodes, and B_{ij} denotes the data transfer rates between c_i and c_j. L is an vector that represent the communication startup costs of computational nodes, and L_i is the communication startup costs of c_i.

3.3 Performance and Energy Consumption Analysis Model

3.3.1 Tasks Scheduling Length

Task scheduling must conform to the precedence-constraints among tasks, and the target processors must be prepared for task execution. The communication cost between task t_i and task t_j is expressed as

$$CC_{ij} = \begin{cases} 0, & k = u, \\ L_k + \frac{data_{ij}}{B_{ku}}, & k \neq u,. \end{cases} \tag{1}$$

where $k = u$ indicates that tasks t_i (scheduled on c_k) and t_j (scheduled on c_u) are assigned to the same processor. Let W_{irk} be the computation cost of task t_i on processor c_k given supply voltage V_{kr}. The earliest execution finish time of task t_i is given by

$$EFT(t_i) = EST(t_i) + W_{ikr}, \tag{2}$$

where $EST(t_i)$ is the earliest execution start time of t_i, which means all immediate predecessor tasks of t_i have been scheduled and the target machine has been ready for its execute, therefore,

$$EST(t_i) = \begin{cases} 0, & pre(v_i) = \varnothing, \\ \max\{avail(c_u) \\ , \max_{t_j \in pre(t_i)}\{EFT(t_j) + CC_{ji}\}\}, & otherwise, . \end{cases} \tag{3}$$

where $avail(c_u)$ is the time at which c_u is ready for the t_i schedule. Finally, tasks scheduling length (also called makespan) is expressed as:

$$makespan(G) = \max_{t_i \in T}\{EFT(t_i)\}. \tag{4}$$

3.3.2 Energy Consumption

In our study, energy consumption analysis is based on the power consumption model in complementary metal-oxide semiconductor logic circuits. Dynamic power dissipation is defined as

$$P = AC'\overline{V}^2 f, \tag{5}$$

where A is the number of switches per clock cycle; C' is the total capacitance load; $\overline{V}$ is the Voltage; and f is the frequency, because $f \propto \overline{V}$, so $P = \varphi\overline{V}^3$, where φ is a parameter that differs with each type processor or machine.

Let the $EB(i)$ be the computation cost of t_i on c_k given supply voltage V_{rk} can be expressed as

$$EB(i) = \varphi_k V_{kr}^3 W_{ikr}, \tag{6}$$

such that the energy consumption of all tasks executed (i.e., busy energy consumption) is determined through

$$EB = \sum_{k=1}^{m} \sum_{t_i \in U_k} \varphi_k V_{kr}^3 W_{ikr}, \tag{7}$$

where U_k is the task set on processor c_k.

The idling processor consumes energy as well and is known as idle power consumption. The idle power consumption of c_k can be calculated as

$$EI(k) = \left(makespan(G) - \sum_{t_i \in U_k} W_{ikr} \right) \varphi_k V_{kg(k)}^3, \tag{8}$$

where $V_{kg(k)}$ is the minimum supply voltage on c_k. Thus, the total idle energy consumption of the system can be defined as

$$EI = \sum_{k=1}^{m} \left(\left(makespan(G) - \sum_{t_i \in U_k} W_{ikr} \right) \varphi_k V_{kg(k)}^3 \right). \tag{9}$$

Therefore, the total energy consumption of a system is expressed as

$$energy(G) = EB + EI. \tag{10}$$

3.4 Problem Model

In [36], the trade-off between energy consumption and the quality of scheduling problems on a homeomorphic multi-processor computer system with dynamically variable voltages is defined as energy-constrained performance optimization ($Pro1$) and performance-constrained energy optimization problem ($Pro2$). By contrast, in this paper, the problem is defined on the HMCSs with VSLs. There are as follows:

1) energy-constrained performance optimization problem ($Pro1$):

$$Minimize : makespan(G), \tag{11}$$

subject to:

$$\begin{gathered} EST(t_i) \geq EFT(t_j), t_j \in pred(t_i), \\ energy(G) \leq Power\_limit. \end{gathered} \tag{12}$$

2) performance-constrained energy optimization problem ($Pro2$):

$$Minimize : energy(G), \tag{13}$$

subject to:

$$\begin{gathered} EST(t_i) \geq EFT(t_j), t_j \in pred(t_i), \\ makespan(G) \leq Time\_limit. \end{gathered} \tag{14}$$

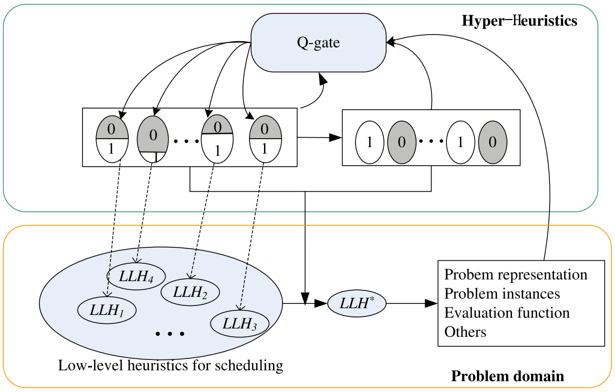


Fig. 2. Quantum-inspired hyper-heuristic framework.

$EST(t_i) \geq EFT(t_j)$, $t_j \in pred(t_i)$ is used to guarantee the scheduled meeting of the precedence-constrains between tasks, $Power\_limit$ and $Time\_limit$ denote maximum energy consumption and task scheduling length, respectively.

4 Methodology

Inspired by quantum computing theory, a quantum-inspired hyper-heuristic framework composed of two levels is proposed in this study (see Fig. 2). At a low level, a reduced heuristics set for scheduling, which has a corresponding expertise, is built. And at a high level, an improved quantum-inspired learning strategy is employed to select promising heuristics and to maintain the diversity of choice. For $Pro1$ and $Pro2$, we only need to change the problem evaluation function in the framework.

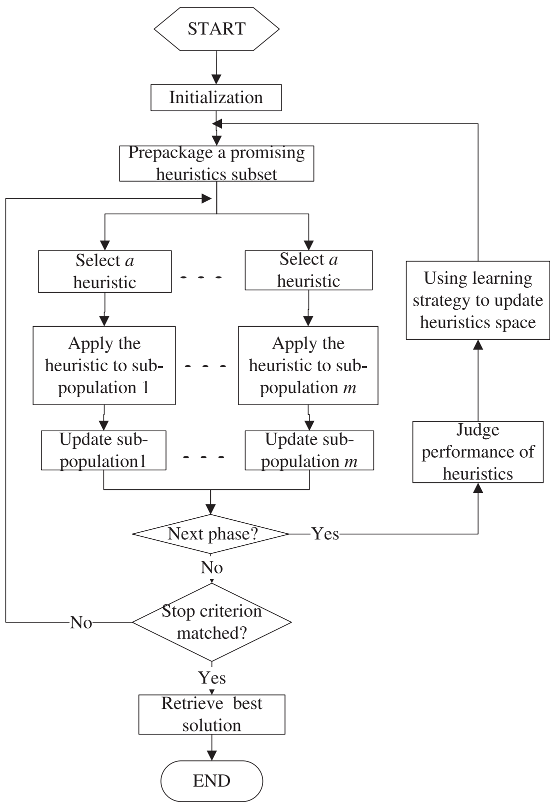


Fig. 3. The process of QHA.

TABLE 2
The Algorithm Parameters and Variables

Symbol	Algorithmic Meaning
$NumP$	The number of sub-populations
SP	The popsize of each sub-populations
s_i	The ith gene of a solution
$stasize$	The number of iterations in a phase
ite	The total iterations number
H	The quantum heuristics state vector
ξ	The number of low-level heuristics
$lh(i)$	The ith low-level heuristic
$pf(i)$	The score of $lh(i)$
$allp$	The score of promising heuristic subset
$Level(t_i)$	The priority of task t_i

The quantum-inspired hyper-heuristic algorithm is a population-based heuristic. Each run starts with initialization, which includes solution and heuristic space initialization. A number of iterations are considered as a phase in the iteration stage. In each phase, an appropriate low-level heuristic is selected from a promising heuristic set to evolve a certain sub-population. After a phase, we update the heuristic space according to a quantum-inspired high-level learning strategy and obtain a promising heuristic set via an observation operation. The stop criterion is based on either the maximum numbers of iterations or the maximum amount of CPU time used. The QHA process is shown in Fig. 3, and the algorithm parameters and variables are presented in Table 2.

4.1 Problem Domain

4.1.1 Coding Scheme and Solution Initialization

Each gene s_i in a solution (Fig. 4) [7] is generally defined by a task, a processor and a voltage level. These three parts of s_i are denoted as $A(s_i)$, $c(s_i)$, and $l(s_i)$, respectively. The search operator on the structure of the solution is usually based on task priority calculation, that is, the task scheduling sequence is normally ordered in advance according to task priority. As a result, parts of the search space may be unreachable. By contrast, our approach, which is illustrated in Fig. 5, offers a solution that is composed of units that correspond to each processor scheduling queues. Furthermore, each gene is defined by a task and a processor. In the search operator, we consider task priority only in the sub-queue; this consideration assists in searching the solution space completely. The priority of task t_i is defined as

$$Level(t_i) = \begin{cases} 0, & t_i = t_{exit}, \\ \max\limits_{t_j \in succ(t_i)} \{Level(t_j)\} + 1, & otherwise. \end{cases} \quad (15)$$

$A(s_1)$	$A(s_2)$	...	$A(s_{n-1})$	$A(s_n)$
$c(s_1)$	$c(s_2)$	...	$c(s_{n-1})$	$c(s_n)$
$l(s_1)$	$l(s_2)$	...	$l(s_{n-1})$	$l(s_n)$

Fig. 4. Illustration of a solution in tradition.

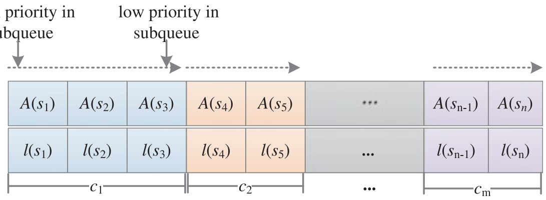


Fig. 5. Illustration of a solution in our approach.

In the solution initialization phase, firstly, the tasks in DAG are sorted topologically, and then are assigned to processor that is randomly selected in turn. $NumP \times SP$ solutions are generated in this manner. The solutions guarantee the precedence-constraint among tasks and ensure the diversity of population.

4.1.2 Reduced Low-Level Heuristics Set

Because the factors (e.g., solution expressions, algorithm parameters) of preexisting heuristics bring much inconvenience to hyper-heuristic application and reduced low-level heuristics are more flexible response to different situation, several reduced heuristics that are specific to DAG scheduling on DVS-enabled HMCSs are applied in this study. Every search step changes only one or two task assignments directly. Tasks move to a new position, and the precedence-constrained among tasks on the same processor must satisfied. The low-level heuristic search strategies are presented in the following list:

1) LLH1. It exchanges the positions of two tasks on different processors with random VSL.
2) LLH2. It moves a task from its current processor to a new processor with random VSL.
3) LLH3. It exchanges the positions of two tasks on the same processor with original VSL.
4) LLH4. It moves a task from its current position to a new position where he communication overhead of the task is minimized.
5) LLH5. It assigns a non-critical task with higher voltage than before.
6) LLH6. It assigns a critical task with less voltage.
7) LLH7. It moves a critical task from its current position to a new position where the computation overhead of the task is minimized.

LLH1, LLH2, and LLH3 have a good global optimization capability to help schedules escape from the local optima.

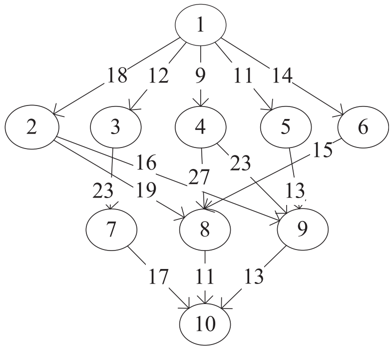


Fig. 6. Sample of DAG with 10 tasks.

LLH4, LLH5, LLH6 and LLH7 are conducive to local optimization based on different local optimization goals.

4.1.3 Fast Solution Evaluation Strategy and Illegal Solution Adjusting

When an individual (i.e. a scheduling) is evaluated, each scheduling task in DAG graph, including the earliest execution start time, earliest execution finish time, and energy consumption, must generally be computed after every search step. This process increase computation cost (e.g., in [6], [7]). If we calculate only the evaluation index of tasks whose status changed(i.e., the task whose earliest execution start or finish time, or energy consumption is altered), much computation overhead is saved. Inspired by this finding, we propose a fast solution evaluation strategy.

As an illustration, Fig. 7 presents a schedule for the sample DAG depicted in Fig. 6, as well as the influence of search operator on the schedule. We can find out such an edge, the status of task on the right side of the edge may be modified, whereas that of the task on the left side of edge does not. To record the edge, an edge search method is presented in Algorithm 1. We adopt a recursive algorithm to record the *edge* with fewest serial numbers on each processor whose task status may change.

A fast evaluation strategy for the makespan evaluation of a solution is highlighted in Algorithm 2. We need only recompute the task whose status may have changed. The solution is adjusted if it does not meet the precedence-constraint, i.e., a task whose pre-tasks have been computed is swapped with the highest prioritized task that has not been computed on the processor. The computation

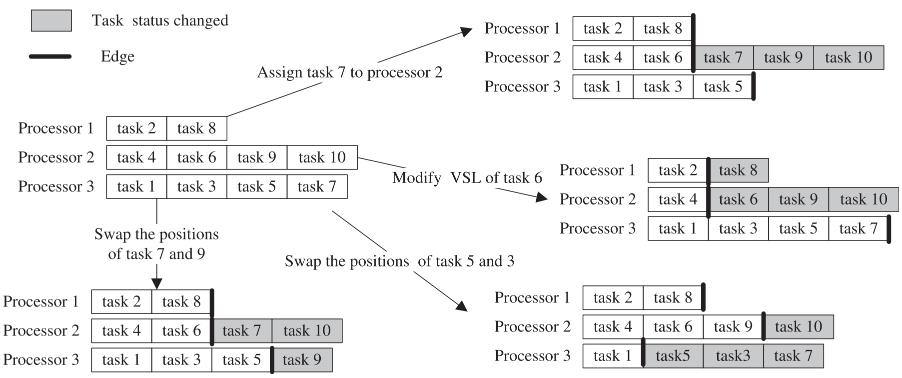


Fig. 7. An example of the influence of basic search operate on tasks.

complexity of correction processing is O(1) and does not significant affect computational efficiency.

Algorithm 1. Edge Search Strategy

$p(t_i)$ is the processor of t_i scheduled on; U_k is a task set on processor k; $SN(t_i)$ is the scheduling serial number of t_i on the processor.
Input: $T\ set$(a task set includes the task whose status changes directly after a search step)
output: $edge$
for all processor k **do**
  $edge(k) = |U_k| + 1$
end for
while $T\ set \neq \emptyset$ **do**
  Select a $t_i \in T\ set$
  if $edge(p(t_i)) > SN(t_i)$ **then**
    $edge(p(t_i)) = SN(t_i)$
  end if
  $T\ set = T\ set \cup \{t_j | t_j \in U_{p(t_i)} \wedge SN(t_i) > SN(t_j) \wedge t_j \notin T\ set\}$
  $T\ set = T\ set \cup \{t_j | t_j \in succs(t_i) \wedge t_j \notin T\ set\}$
  Remove t_i from $T\ set$
end while

Algorithm 2. Fast Solution Evaluation Strategy

CT is the vector record serial number of computing tasks currently being on each processors with an initial value of zero; $US(k)$ is the start position of unit k in the solution; $NP(t_i)$ records the number of immediate predecessor of task t_i which not be computed.
Input: $edge$ and solution s
output: $makespan(G)$
$i = 0$
while $i \leq N$ **do**
  $fig = 0$
  for $k = 1$ to m **do**
    if $CT(k) < |U_k|$+1 and $NP(A(s_{US(k)+CT(k)})) = 0$ **then**
      $CT(k) = CT(k) + 1$
      $NP(t_j) = NP(t_j) - 1, \forall t_j \in succ(A(s_{US(k)+CT(k)}))$
      if $CT(k) > edge(k)$ **then**
        Recompute $EST(A(s_{US(k)+CT(k)}))$ and $EFT(A(s_{US(k)+CT(k)}))$
      end if
      $fig = 1$
      $i = i + 1$
    end if
  end for
  if $fig = 0$ and $i <$ N **then**
    Search a task t_j which $NP(t_j) = 0$
    Swap $A(s_{CT(p(t_j))})$ and t_j
  end if
end while
$makespan(G) = \max_{t_i \in T}\{EFT(t_i)\}$

4.1.4 Constraint Handing and Solution Selection

In our approach, a constraint relaxation strategy is adopted for the solution that does not satisfy the constraints of energy consumption or makespan (i.e. infeasible solution). When two solutions are compared at a time, the following criteria are enforced [46], [47].

1) The feasible solution is preferred to the infeasible solution.
2) Among two feasible solutions, the one having better objective function value is preferred.
3) Among two infeasible solutions, the one having smaller violation is preferred.

4.2 Quantum Inspired High-Level Heuristic

The high-level heuristic of hyper-heuristic framework is developed to select an appropriate heuristic to search the solution space. The performance of heuristic may change dynamically in different search phase. Therefore, the high-level heuristic should be able to choose the most appropriate heuristic quickly and have the capability to escape the phase optimal. To achieve it, we require a mechanism that can track the performance of low-level heuristics in real time or phase, that is, the selected probability of low-level heuristics is updated based on its real-time or phase performance.

In quantum computing [48], Q-bit is the smallest unit that can specify the probability amplitudes of the corresponding states. The state of a Q-bit can be changed to a more reasonable state by Q-gate, which is similar to the selected heuristic probability update process. Moreover, our previous study [49], [50] show that the quantum-inspired evolution algorithm exhibits good global optimization and dynamic adaptability performance. thus, a quantum inspired high-level heuristic is proposed in current work.

4.2.1 Heuristics Search Space Expression and Initialization

In our quantum inspired hyper-heuristic framework, a heuristic Q-bit $h = [\begin{smallmatrix}\alpha \\ \beta\end{smallmatrix}]$ corresponds to a low-level heuristic, $|\alpha|^2$ gives the probability that the heuristic disuse states and $|\beta|^2$ presents the probability that the heuristic selects state, α and β guarantees $|\alpha|^2 + |\beta|^2 = 1$. A heuristic Q-bit may be in the "0" state (the heuristic is not selected) and in the "1" state (heuristic is selected) after observation. A heuristics search space with ξ low-level heuristics is defined as $H = (h_1, h_2, \ldots, h_\xi) = \begin{bmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_\xi \\ \beta_1 & \beta_2 & \cdots & \beta_\xi\end{bmatrix}$.

In the heuristics search space initialization, α_i and β_i, $i = 1, 2, \ldots, \xi$, are initialized with $1/\sqrt{2}$, thus suggesting that all heuristics have the same selected or unselected probabilities.

4.2.2 Learning Strategy and Heuristic Selection

A learning strategy based on Q-gate is used to update the heuristic space after a phase is completed. The rotation gate $U(\Delta\theta_i)$ [31] is used as a Q-gate, which is defined as

$$\begin{bmatrix}\alpha'_i \\ \beta'_i\end{bmatrix} = \begin{bmatrix}cos(\Delta\theta_i) & -sin(\Delta\theta_i) \\ sin(\Delta\theta_i) & cos(\Delta\theta_i)\end{bmatrix}\begin{bmatrix}\alpha_i \\ \beta_i\end{bmatrix}, \tag{16}$$

where $\Delta\theta_i$ is a rotation angle of each Q-bit toward either 0 or 1 state. $\Delta\theta_i$ is determined with a lookup table in [31], which ignore the difference of Q-bit performance. By contrast, $\Delta\theta_i$ adapts to changes in our study according to the ith heuristic performance in a promising heuristic subset. The $\Delta\theta_i$ is defined as

$$\Delta\theta_i = \begin{cases} 0, & lh(i) \notin S, \\ \frac{pf(i)-\frac{1}{|S|}\sum_{lh(i)\in S} pf(i)}{Q\times Ran}\pi, & lh(i)\in S, |S| > 1, \\ \frac{pf(i)-\gamma}{Q}\times\pi, & lh(i)\in S, |S| = 1, \end{cases} \quad (17)$$

where $lh(i)$ is the ith low-level heuristic; S is a heuristic set selected to search the solution space in phases; and $|S|$ is the size of the heuristic set. $pf(i)$ is the score of $lh(i)$. If the ith heuristic in a phase is applied, $pf(i)$ is equal to the rate of the solutions' improvement times (based on the criterion introduced in Section 4.1.4) to the heuristic applied times. Otherwise, $pf(i) = 0$. $Ran = \max\{pf\}$-$\min\{pf\}+\varepsilon$, where ε is a small positive number that ensures that the denominator is not equal to zero. γ $(0 < \gamma < 1)$ is a parameter that can be set up in advance or produced at random. Q is a constant guaranteeing that the $\Delta\theta_i$ range is $[-0.05\pi, 0.05\pi]$.Therefore, this range meets the experience value range in [31]. The Eqn. (17) generates the heuristic Q-bit associated with heuristic performance.

The learning strategy is shown in Algorithm 3. Firstly, we update each Q-bits based on $\Delta\theta_i$, which is in turn associated with the ith heuristic performance in the phase. If the average score of applied heuristics (i.e., $allp$) is less than a threshold value σ, then a randomly generated angle parameter λ $(0.01\pi \leq \lambda \leq 0.05\pi)$ is used to reduce the selected heuristic probability of state "1" and increase the unselected heuristic probability of state "1."

Algorithm 3. Quantum-Inspired High-Level Learning Strategy

Input: pf, $allp$, H
Output: H
$i \leftarrow 1$
while $i \leq \xi$ **do**
  Obtain $\Delta\theta_i$ by Eqn. (17)
  if h_i is located in the first / third quadrant **then**
    $[\alpha_i{}'\beta_i{}']^T = \mathrm{U}(-\Delta\theta_i)[\alpha_i\beta_i]^T$
  else
    $[\alpha_i{}'\beta_i{}']^T = \mathrm{U}(\Delta\theta_i)[\alpha_i\beta_i]^T$
  end if
  $i \leftarrow i+1$
end while
$H' \leftarrow H$
if $allp < \sigma$ **then**
  $i \leftarrow 1$
  while $i \leq \xi$ **do**
    $\Delta\theta_i' = \lambda$
    if ith low-level heuristic is applied **then**
      Using $\Delta\theta_i'$ to reduce the h_i' probability of the state "1"
    else
      Using $\Delta\theta_i'$ to increase the h_i' probability of the state "1"
    end if
    $i \leftarrow i+1$
  end while
end if
$H \leftarrow H'$

In each phase of our approach, a promising heuristic subset is obtained by observing H. The observation process is presented in Algorithm 4. In a Q-bit observation, a random value is generated from the range [0], [1]. If the value is less than $|\beta_i|^2$, then the ith heuristic is selected to promising heuristic subset. If no heuristic is selected during the above observations, then a random heuristic in the set of low-level heuristics is selected to promising heuristic subset.

TABLE 3
Setting of Algorithm Parameters

Parameter	value
$NumP$	3
SP	10
$stasize$	20
ite	1,500
σ	$1/20$

Algorithm 4. Obtain Promising Low-level Heuristic

$Subset\_H$ is the promising low-level heuristic subset, and lh is a low-level heuristic vector
Input: H,lh, ξ
Output: $subset\_H$
$i \leftarrow 1$
$Subset\_H \leftarrow \emptyset$
  while $i \leq \xi$ **do**
    if rand()$\leq \beta_i^2$ **then**
      $Subset\_H = Subset\_H \cup \{lh(i)\}$
    end if
  end while
  if $Subset\_H = \emptyset$ **then**
    $Subset\_H \leftarrow$ rand(lh)
end if

In every iteration, a heuristic from the promising heuristic subset chosen by roulette selection, is applied to a certain sub-population. In the roulette selection, the selection probability of the ith heuristic is written as $hp(i) = \frac{\beta_i^2}{\sum_{i=1}^{\kappa}\beta_i^2}$, where κ is the number of heuristics in promising heuristic set.

5 Experiment Results and Discussion

5.1 Experiment Environment

The simulation experiments were performed on a PC running on an Intel Pentium 2.70 GHz CPU and a 2 GB RAM. The experimental tool is MATLAB R2009a, and the algorithm parameters are set as in Table 3.

5.1.1 Heterogeneous System Simulation

The target system in the simulation environment comprises a set of fully interconnected heterogeneous processors. Every system processor is DVS-enabled, i.e., the processor can operate in VSLs, and various VSLs corresponds to different computation speeds of processor. In each experiment, a certain number of processors are randomly and uniformly distributed among four different sets of VSLs (Table 4) [6]. Moreover, the φ_k value of processor c_k is randomly generated from the range [14], [16].

5.1.2 DAG Sets

We consider two sets of graphs to test the algorithms. The graphs represent the real-world problems (modified

TABLE 4
Voltage-Relative Speed Pairs

level	pair1		pair2		pair3		pair4	
	voltage	Relative speed(%)	voltage	Relative speed(%)	voltage	Relative speed(%)	voltage	Relative speed(%)
0	1.75	100	1.5	100	2.2	100	1.5	100
1	1.4	80	1.4	90	1.9	85	1.2	80
2	1.2	60	1.3	80	1.6	65	0.9	50
3	0.9	40	1.2	70	1.3	50		
4			1.1	60	1	35		
5			1	50				
6			0.9	40				

molecular dynamic code [51]) and randomly generated application graphs. The random graph generator improves on that presented in literature [21] in that the previous graph generator did not consider system DVS. The input parameters of our random graph generator are listed in Table 5. In generating a random DAG, the number of tasks in a graph (i.e., N) is provided in advance, and the graph height is randomly generated from a uniform distribution with a mean value equal to $\frac{\sqrt{N}}{\alpha}$. A larger α value indicates high parallelism. The minimum computation cost (i.e., the computation cost incurred when all the processors supply with their maximum voltage) of each task t_i on each processor c_k, i.e., $minw_{ik}$, is set at random on the basis of range $[\overline{w_i} \times (1-\frac{\delta}{2}),\ \overline{w_i} \times (1+\frac{\delta}{2})]$, where δ is the range percentage of computation costs on processors. A large δ value causes a significant difference in the computation costs of tasks among the processors. $\overline{w_i}$ is the average computation cost of each task t_i, which is selected randomly from a uniform distribution with a range of $[0,\ 2 \times \overline{w_{DAG}}]$($\overline{w_{DAG}}$ is the average computation cost of the given DAG). The computational cost of each task t_i on each processor c_k with supply voltage V_{kr}, in which the relative speed is $Rs(V_{kr})$, i.e., W_{ikr}, is set as $\frac{minw_{ik}}{Rs(V_{kr})}$. The communication cost among tasks is selected randomly from a uniform distribution with range $[0, 2\times \overline{w_{DAG}} \times CCR]$. CCR is communication to computation ratio. A low CCR value in DAG indicates a computation-intensive application.

5.2 Comparison Metrics

The energy consumption and makespan of a task graph as generated by a scheduling algorithm is used as a performance measure. For a given task graph, we normalize the energy consumption of the tasks to a lower bound, which is called the energy consumption ratio (ECR), the ECR value of an algorithm on a task graph is expressed as:

$$ECR = \frac{energy}{\sum_{i=1}^{n} \min_{c_k \in C}\{\varphi_k \times W_{ikg(k)} \times V_{kg(k)}^3\}}. \tag{18}$$

The denominator of ECR is the summation of the minimum energy computation for each task. The task scheduling algorithm that generates the lowest ECR for a graph is the best algorithm with respect to most energy conservation.

Schedule length ratio (SLR) [21] is another comparison metric, the SLR of an algorithm on a task graph is defined as

$$SLR = \frac{makespan}{\sum_{t_i \in P_{MIN}} \min_{c_k \in C}\{W_{ikr}\}}, \tag{19}$$

where the ideal task scheduling algorithm with respect to performance is one with low SLR value for a graph.

5.3 Performance of Fast Solution Evaluation Strategy

The first set experiments compares the computing cost of traditional solution evaluation and our fast solution evaluation strategy. The performance of the latter was examined given various graph sizes (Fig. 8). In Fig. 8, $T1 = (\sum_{i=1}^{7} t1_i)/7$, where $t1_i$ is the average complete computing time of the ith low-level heuristic under 1,000 evaluations for 50 runs when the traditional solution evaluation method is used. $T2 = (\sum_{i=1}^{7} t2_i)/7$, where $t2_i$ is the average complete computing time of the ith low-level heuristic under 1,000 evaluations for 50 runs when the fast evaluation method is applied. Given various graph sizes, the fast evaluation strategy is faster than traditional solution evaluation method by 36.35, 38.03, 39.86, 39.34 and 38.29 percent

TABLE 5
Input Parameters of Random Graph Generator

Parameters	Values	Introduction
N	(16,32,64,128,256)	The number of tasks in the graph
α	(0.5,1.0,2)	The shape parameter of the graph
δ	(0.1,0.25,0.5,0.75,1)	The range percentage of computation costs
CCR	(0.1,0.5,1,2,5,10)	The communication to computation ratio

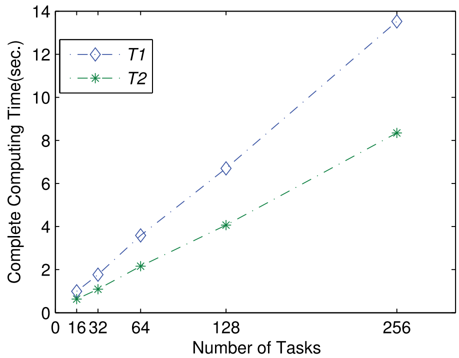


Fig. 8. Average complete computing time obtain with different solution evaluation strategies respect to graph size.

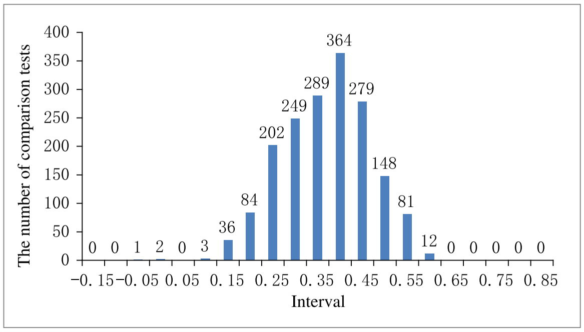


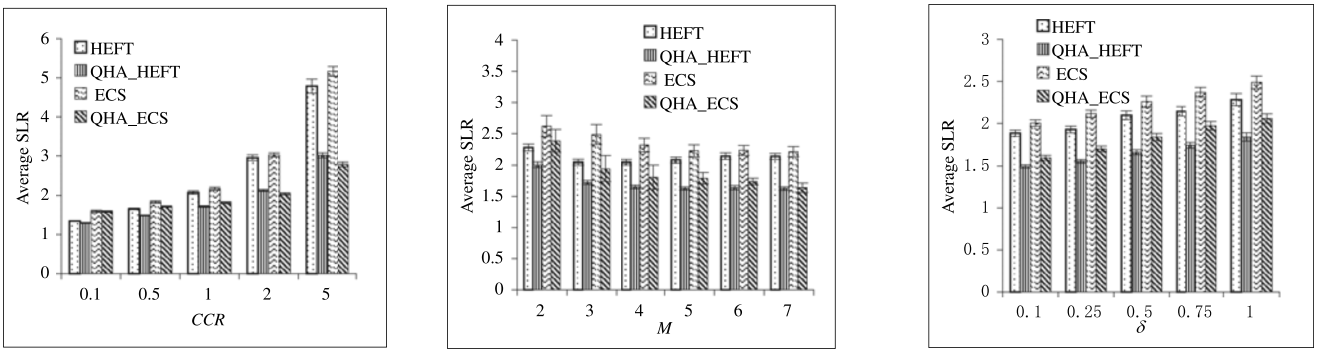
Fig. 9. The number of comparison tests in each improvement rate interval.

respectively. We also record the number of comparison tests in each improvement rate interval in 1,750 ($5 \times 7 \times 50$) comparison tests (Fig. 9). It shows that our approach can improve the computation speed of an algorithm by more than 25 percent in 93 percent comparison tests.

5.4 Application Graphs of Real Word Problems

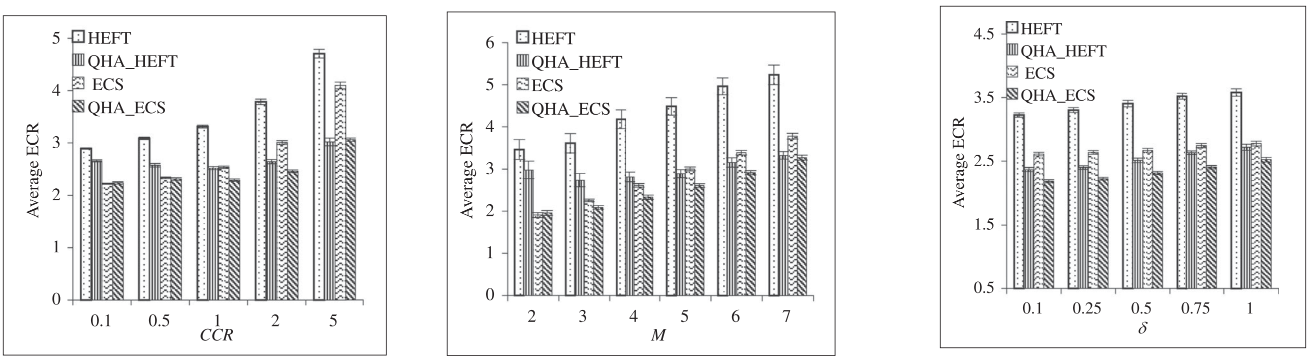
In our study, we considered application graphs of real-world problems, namely, modified molecular dynamic code [51]. QHA parameters are set as shown in Table 3. In the experiment process, the results of QHA are compared with those of the state-of-the-art list algorithm (HEFT [21]) and energy-aware scheduling algorithm (ECS [6]). The values of CCR, δ, M (in Table 5) are applied. A total of 50 independent runs are performed for each set of contrast test. In each run, we perform HEFT and ECS simulations. Then the $energy(G)$ and $makespan(G)$ obtained by HEFT and ECS is applied as $Pro1$'s energy-constraint, and $Pro2$'s performance-constraint, respectively. The result of QHA simulation is constrained by output of HEFT is known as QHA_HEFT. The result constrained with the output of ECS is known as QHA-_ECS. For different problems ($Pro1$ or $Pro2$), QHA alters only the problem evaluation function of the framework.

Fig. 10 depicts the average result for $Pro1$ with 95 percent confidence intervals. Fig. 10a displays the result of algorithms with respect to different CCR values. On the average, the SLR ranking(from small to large) is {QHA, HEFT, ECS}. Figs. 10b and 10c exhibit the SLR value of algorithms given different processor number(M) and δ values, respectively. QHA outperform HEFT and ECS, and HEFT is slightly superior to ECS. The average SLR value of QHA on all graphs is less than those of HEFT and ECS algorithms by 19.4 and 19.6 percent, respectively. Fig. 11 presents the average ECR value of $Pro2$ with 95 percent confidence intervals. Specifically, Figs. 11a, 11b, and 12c show the results on DAG with different SLR , M and δ values, respectively. On the average, the ECR ranking is {QHA, ECS, HEFT}. The average ECR value of QHA on all graphs is better than those of HEFT and ECS algorithms by 26.5 and 11.3 percent, respectively. Based on the analysis above, The HEFT algorithm generates a better SLR value than ECS algorithm does for performance optimization. However, the ECS



(a) (b) (c)

Fig. 10. Average result for $Pro1$ with 95 percent confidence intervals. (a) Result of the algorithms with respect to different CCR values when $M = 4$ and $\delta = 0.5$. (b) Result of the algorithms with respect to different M values when $CCR = 1.0$ and $\delta = 0.5$. (c) Result of the algorithms with respect to different δ values when $CCR = 1.0$ and $M = 4$.



(a) (b) (c)

Fig. 11. Average result for $Pro2$ with 95 percent confidence intervals. (a) Result of the algorithms with respect to different CCR values when $M = 4$ and $\delta = 0.5$. (b) Result of the algorithms with respect to different M values when $CCR = 1.0$ and $\delta = 0.5$. (c) Result of the algorithms with respect to different δ values when $CCR = 1.0$ and $M = 4$.

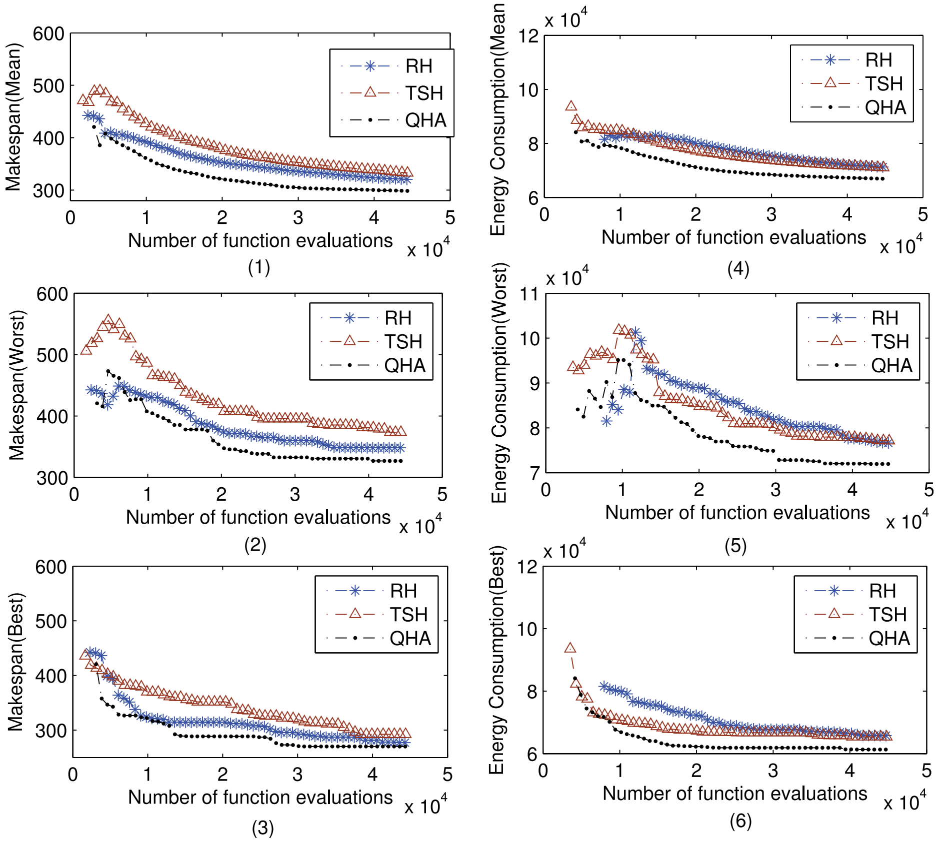


Fig. 12. Plots versus the duration of a simulation rum of D11.(1), (3), (5) Mean, Worst, Best simulate result for $Pro1$, respectively.(2), (4), (6) Mean, worst, best simulate result for $Pro2$.

algorithm reports a lower ECR value than the HEFT algorithm does. Nonetheless, the values obtained with QHA are always the best among the three algorithms because QHA displays an outstanding universal performance. By contrast, constructive heuristic scheduling algorithm (HEFT and ECS) apply greedy local optimization to a certain optimization goal, don't have global optimization capability.

The average of computation time (sec.) of HEFT, ECS and QHA is 0.08, 5.14 and 52.70, respectively. We have to admit that, the running time of QHA remains higher than those of the HEFT and ECS, although the computation cost of QHA is considerably lower than those of other algorithms based on search techniques. Nonetheless, QHA exhibits good parallelism, and its running times can be reduced further using parallel programming technology.

5.5 Search Efficiency of QHA

We test the search effectiveness of QHA with 18 DAGs instances (Table 6) from random DAG generator. These instances consider the influence of different application graphs and the number of processors. The results of QHA are compared with three state-of-the-art algorithms, namely, GA for multiprocessor scheduling [23], and a rand hyper-heuristics (RH), which are often compared by papers on hyper-heuristic, and the tabu-search hyper-heuristic (TSH) [37]. These three algorithms are with same constraints-handling techniques (Section 4.1.4) and are implemented according to papers in matlab language. We compare the results within a certain number of function evaluations, which is set as 45,000 in the experiment. QHA parameters are set as shown in Table 3. Moreover, the population size, crossover and mutation probability of GA are set as 20, 0.5 and 0.005, respectively. The population size, learning step, r_{min} and r_{max} value of TSH, are set as 30, 1, 0 and ξ, respectively.

We obtained the statistics by repeating the simulation run 50 times for each instances on $Pro1$ and $Pro2$. The "Feasible rate," "Best," "Worst," "Mean" and "std" of each set of data represent the ratio of feasible solution, the best case, the worst case, the average and the standard deviation

TABLE 6
Test Instance

instance	N	M	α	CCR	δ	$Time\_limit$	$Energy\_limit$
D1	16	8	1	1	0.5	80	12,000
D2	32	8	1	1	0.5	150	25,000
D3	64	8	1	1	0.5	250	35,000
D4	128	8	1	1	0.5	300	70,000
D5	256	8	1	1	0.5	700	140,000
D6	128	8	0.5	1	0.5	600	90,000
D7	128	8	2	1	0.5	250	60,000
D8	128	8	1	1	0.1	300	65,000
D9	128	8	1	1	0.25	300	70,000
D10	128	8	1	1	0.75	300	68,000
D11	128	8	1	1	1	300	68,000
P12	128	2	1	1	0.5	800	60,000
P13	128	4	1	1	0.5	550	60,000
P14	128	16	1	1	0.5	250	90,000
D15	128	8	1	0.1	0.5	300	60,000
D16	128	8	1	0.5	0.5	300	68,000
D17	128	8	1	2	0.5	400	80,000
D18	128	8	1	10	0.5	800	140,000

TABLE 7
Result of $Pro1$

	instance	D1	D2	D3	D4	D5	D6	D7	D8	D9
GA	Feasible rate (%)	0	98	0	12	0	0	0	0	0
	Worst	*	*	*	*	*	*	*	*	*
	Best	*	2.1027	*	3.6701	*	*	*	*	*
	Mean	*	*	*	*	*	*	*	*	*
	std	*	*	*	*	*	*	*	*	*
RH	Feasible rate (%)	66	100	100	100	100	100	100	100	100
	Worst	*	2.0274	2.6421	3.4468	4.0097	2.3632	5.1828	3.1774	3.4811
	Best	1.7039	1.6164	2.2737	3.0552	3.5809	2.0602	4.6157	2.7643	3.0882
	Mean	*	1.7991	2.4584	3.2503	3.7784	2.2439	4.8827	2.9820	3.2989
	std	*	7.64E-02	6.99E-02	1.13E-01	8.45E-02	5.49E-02	1.28E-01	9.17E-02	8.65E-02
TSH	Feasible rate (%)	58	100	100	100	100	100	100	100	100
	Worst	*	1.9909	2.8759	3.6727	4.3468	2.5653	5.4813	3.3666	3.7185
	Best	1.6818	1.6849	2.3579	3.0526	3.7503	2.2293	4.7239	2.9653	3.2395
	Mean	*	1.8279	2.5839	3.4266	4.1255	2.3985	5.1291	3.1776	3.4689
	std	*	7.31E-02	9.59E-02	1.20E-01	1.37E-01	8.21E-02	1.84E-01	1.03E-01	1.12E-01
QHA	Feasible rate (%)	**74**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	Worst	*	**1.9178**	2.6465	3.5234	**3.9513**	2.5683	**5.0448**	3.2643	3.4958
	Best	**1.5818**	**1.5479**	**2.1684**	**2.8610**	**3.3300**	**2.0536**	**4.3433**	**2.6107**	**2.8676**
	Mean	*	**1.7549**	**2.4425**	**3.1549**	**3.5973**	**2.2390**	**4.7817**	3.0052	**3.2528**
	std	*	**7.12E-02**	1.11E-01	1.30E-01	1.48E-01	7.55E-02	**1.23E-01**	1.27E-01	1.29E-01

	instance	D10	D11	P12	P13	P14	D15	D16	D17	D18
GA	Feasible rate (%)	2	76	8	2	0	0	24	54	0
	Worst	*	*	*	*	*	*	*	*	*
	Best	4.8328	4.6325	10.1081	5.9985	*	*	3.5003	4.4839	*
	Mean	*	*	*	*	*	*	*	*	*
	std	*	*	*	*	*	*	*	*	*
RH	Feasible rate (%)	100	100	100	100	84	100	100	100	100
	Worst	4.5103	4.2078	10.0290	5.8668	*	3.6912	3.2727	3.8403	7.7955
	Best	3.8011	3.4518	9.4277	5.0460	2.7350	2.8664	2.7455	3.3614	6.2000
	Mean	4.1146	3.8842	9.7383	5.4613	*	3.2690	3.0439	3.5667	7.1023
	std	1.42E-01	1.74E-01	1.30E-01	1.64E-01	*	1.23E-01	1.24E-01	1.14E-01	3.26E-01
TSH	Feasible rate (%)	100	100	100	100	43	100	100	100	100
	Worst	4.8295	4.3348	11.2173	6.2593	*	3.8636	3.5818	3.8857	7.2727
	Best	3.9034	3.6274	9.8920	5.2850	3.0664	3.2670	2.9636	3.3182	6.2455
	Mean	4.3314	3.9896	10.6284	5.9097	*	3.6207	3.2273	3.6423	6.8345
	std	1.81E-01	1.80E-01	3.29E-01	1.79E-01	*	1.50E-01	1.34E-01	1.45E-01	2.64E-01
QHA	Feasible rate (%)	**100**	**100**	**100**	**100**	66	**100**	**100**	**100**	**100**
	Worst	**4.3439**	**3.8898**	10.0849	**5.7469**	*	**3.5740**	**3.1523**	**3.6402**	**7.2409**
	Best	**3.6563**	**3.2108**	**9.1754**	5.0584	*	2.9003	**2.7414**	**3.1045**	**5.7909**
	Mean	**3.9802**	**3.6305**	**9.7037**	**5.3849**	**2.6024**	3.3191	**2.9429**	**3.3543**	**6.4890**
	std	**1.38E-01**	**1.71E-01**	1.93E-01	**1.49E-01**	*	1.30E-01	**1.06E-01**	**1.13E-01**	3.27E-01

of 50 runs, respectively. The values of QHA in the black body indicate that QHA generates the best solution among all meta-heuristics. “*” indicates that the data are infeasible. Tables 7 and 8 present the simulation results of algorithms for $Pro1$ and $Pro2$, respectively.

Table 7 indicates that QHA outperform the other three algorithms in many of the test instances, such as D1, D2, D7, D10, D11, D16, D17. By contrast, the other three algorithms cannot outperform the QHA in any instance. The same situation is shown in Table 8; QHA performs better in D4, D5, D7, D8, D9, D10, P14, and D16 than the other three meta-heuristics. Similarly, the latter cannot outperform QHA in any instance. In addition, QHA has a higher Feasible rate and Mean values than the others Meta-heuristics in almost all of instances in Table 8. Furthermore, GA cannot find a feasible solution in some instances and performed poorly both for $Pro1$ P and $Pro2$, as per these tables.

$Pro1$ and $Pro2$ are optimization problems that are subject to constraint, demanding higher search efficiency for algorithms. The traditional algorithm GA performs poorly despite the inclusion of a relaxation strategy. Moreover, RH performs better than GA, which indicates the reduced low-level heuristic set, in which heuristic change only one or two task assignments directly in every search step, is feasible. And a complicated search operator may not achieve better performance than the reduced operator in some case. Moreover, QHA outperform other two hyper-heuristic (RH and TSH) indicate that the Quantum-inspired high-level strategy is effective.

To determine the performance of QHA during a simulation run, we examine the D11 case for $Pro1$ and $Pro2$ in detail. We plot the results versus the number of function evaluations during a simulation, as presented in Fig. 12. We record the best result after each 750 function evaluation intervals. Each data point in all the plots in Fig. 12 represents a statistical result (Mean, Best, and Worst) in 50 runs at a time instance. Fig. 12 (1, 3, 5) shows the plots of simulated result for $Pro1$, and Fig. 12 (2, 4, 6) presents the

TABLE 8
Result of $Pro2$

	instance	D1	D2	D 3	D4	D5	D6	D7	D8	D9
GA	Feasible rate	0	0	28	0	0	0	0	0	0
	Worst	*	*	*	*	*	*	*	*	*
	Best	*	*	3.3907	*	*	*	*	*	*
	Mean	*	*	*	*	*	*	*	*	*
	std	*	*	*	*	*	*	*	*	*
RH	Feasible rate	94	100	100	100	100	100	0.94	62	62
	Worst	*	3.1310	2.7182	3.4296	3.2758	3.5908	3.6178	3.6790	3.5731
	Best	3.4955	2.6634	2.4072	3.0520	2.9425	3.3266	2.8882	3.0437	3.0353
	Mean	*	2.9596	2.6068	3.2367	3.1247	3.4564	3.2631	3.2179	3.2833
	std	*	7.76E-02	7.07E-02	8.42E-02	7.48E-02	6.26E-02	1.63E-01	1.68E-01	1.34E-01
TSH	Feasible rate	96	100	100	100	100	100	88	73	84
	Worst	*	3.0934	2.8322	3.5243	3.2611	3.6178	3.6781	3.6049	3.5784
	Best	3.6847	2.6823	2.3529	2.9955	2.9287	3.3202	2.9718	2.9318	3.1725
	Mean	*	2.9327	2.6165	3.2412	3.0946	3.4724	3.3151	3.1890	3.3268
	std	*	8.58E-02	8.93E-02	1.16E-01	7.53E-02	5.80E-02	1.76E-01	1.74E-01	1.18E-01
QHA	Feasible rate	**98**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	Worst	*	**3.0348**	2.7727	**3.2542**	**3.0942**	3.6484	**3.5674**	**3.5145**	**3.4857**
	Best	3.5101	2.7566	2.4560	**2.9061**	**2.7793**	**3.3183**	**2.8826**	**2.8698**	**2.8662**
	Mean	*	**2.9083**	**2.5960**	**3.0641**	**2.9090**	**3.4487**	**3.1661**	**3.0415**	**3.1109**
	std	*	**7.10E-02**	**6.71E-02**	**1.01E-01**	**6.99E-02**	7.48E-02	**1.41E-01**	**1.31E-01**	1.41E-01

	instance	D10	D11	P12	P13	P14	D15	D16	D17	D18
GA	Feasible rate	0	0	0	8	0	0	0	0	0
	Worst	*	*	*	*	*	*	*	*	*
	Best	*	*	*	2.6874	*	*	*	*	*
	Mean	*	*	*	*	*	*	*	*	*
	std	*	*	*	*	*	*	*	*	*
RH	Feasible rate	100	100	100	100	100	100	100	100	0.46
	Worst	3.8453	4.0803	3.1115	2.5398	4.3219	2.9903	3.6231	3.7021	5.7285
	Best	3.1825	3.3979	2.8654	2.1500	3.7387	2.5275	2.7522	2.7085	4.7992
	Mean	3.5401	3.7609	3.0185	2.3566	4.0899	2.7915	3.1085	3.2189	5.2055
	std	1.38E-01	1.68E-01	5.91E-02	7.09E-02	1.25E-01	9.60E-02	2.07E-01	2.40E-01	3.03E-01
TSH	Feasible rate	100	100	100	100	100	100	100	100	89
	Worst	3.7157	4.0615	3.2251	2.7045	4.4163	3.0879	3.3985	3.5126	5.7617
	Best	3.1344	3.3235	2.5976	2.2757	3.7287	2.6728	2.7743	2.7896	4.3921
	Mean	3.4942	3.7191	2.9248	2.5008	4.0420	2.8375	3.0670	3.1196	4.9868
	std	1.39E-01	1.49E-01	1.50E-01	9.92E-02	1.31E-01	9.76E-02	1.45E-01	1.62E-01	3.37E-01
QHA	Feasible rate	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	Worst	**3.6129**	**3.8498**	**3.1232**	**2.4715**	**4.1623**	**2.9087**	**3.1323**	**3.3580**	**5.4854**
	Best	**3.0821**	**3.2029**	**2.8530**	**2.1058**	**3.6619**	2.5678	**2.6130**	2.7605	4.5506
	Mean	**3.3192**	**3.5319**	2.9846	**2.3408**	**3.9494**	**2.7675**	**2.8985**	**3.0508**	**4.9148**
	std	**1.15E-01**	1.57E-01	6.33E-02	7.73E-02	**1.23E-01**	**8.35E-02**	**1.24E-01**	**1.36E-01**	**2.25E-01**

simulated result of $Pro2$. We can observe that QHA always obtained better results than the other algorithms.

The selection probability of each low-level heuristic in different phases for QHA is shown in Fig. 13. For all the plots in Fig. 13, each data point represents the mean of 50 runs at a time instance. Fig. 13a presents the result for $Pro1$. The selection probability of each heuristic varies according to different phases. In subsequent phases, the

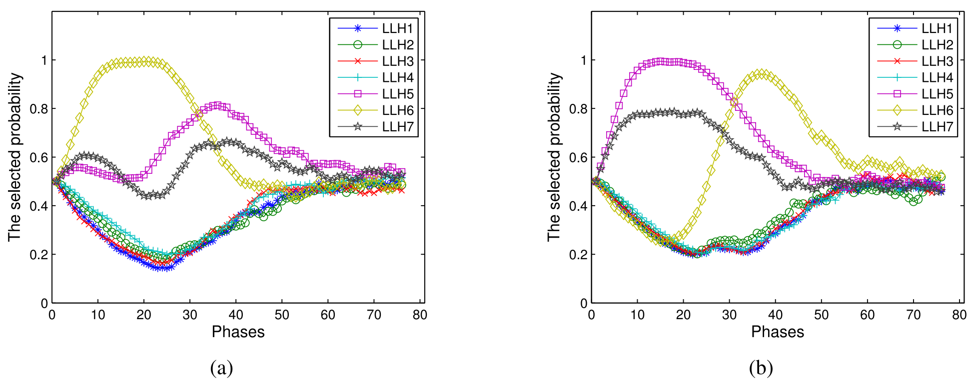


Fig. 13. Selected probability of low-level heuristics in different phases. (a) Result of $Pro1$. (b) Result of $Pro2$.

selected probability for all heuristic is close to 0.5. The same phenomenon is observed for *Pro*2 in Fig. 13b because dissimilar heuristics perform differently in various phases; for example, for *Pro*2, the algorithm must reduce the solution makespan to identify the feasible solution in the early phases. LLH5 performs effectively in this area. After searching in the feasible area, the algorithm focus on optimizing energy consumption, and LLH6 performs well in this area, thus obtaining a higher selection probability. In the later stage of optimization, each low-level heuristic has less probability of improving the solution. The selected probability fluctuates at approximately 0.5. The selected probability of the same heuristic in the same phases also differs from those displayed in Figs. 13a and 13b because the performance of heuristics for different problems also differs. Overall, the Quantum-inspired high-level learning strategy could obtain an accurate response from the current performance of the low-level heuristic while maintaining the diversity of choice.

6 Conclusion

The power and performance tradeoff optimization on HMCSs is an important issue that must be addressed urgently. In this paper, an improved hyper-heuristic, QHA, is proposed for energy-aware scheduling on HMCSs with VSLs. First, the problem is defined as *Pro*1 and *Pro*2 to manage power consumption flexibly in HMCSs. In addition, a hyper-heuristic framework is introduced to improve the universality of scheduling algorithm. Moreover, a fast solution evaluation technique and quantum-inspired high-level strategy are development to reduce the computational cost and to improve search efficiency of algorithm. Furthermore, a reduced low-level heuristic set is established. The simulation results and discussion indicate that the proposed method exhibits excellent performance compared with state-of-the-art methods (HEFT, ECS, GA, RH and TSH).

Although our method presents a large improvement compared with previous approaches based on guided random search technique, highly optimized computing time is also needed. In future works, we will improve the low-level heuristic set and heuristic state update mechanism to enhance algorithm performance. Moreover, we will introduce parallel computing to shorten the execution time because QHA displays high-level parallelism.

Acknowledgments

The work reported in this paper was supported by the National Natural Science Foundation of China (Grant No. 61173107), the National High Technology Research and Development Program of China (Grant No. 2012AA01A301-01), the Special Project on the Integration of Industry, Education and Research of Guangdong Province, China (No.2012A090300003) and the Science and Technology Planning Project of Guangdong Province, China (No. 2013B090700003). Zhiyong Li is the corresponding author.

References

[1] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor Real-time systems[J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 7, pp. 686–700, Jul. 2003.

[2] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters[C], in *Proc. ACM/IEEE Comput. Soc. Conf. Supercomput.*, 2005, p. 34.

[3] D. P. Bunde, "Power-aware scheduling for makespan and flow," in *Proc. 18th Ann. ACM Symp. Parallelism Algorithms Archit.*, Jul. 2006, pp. 190–196.

[4] K. Li, "Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed[J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1484–149, Nov. 2008.

[5] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters [J]," *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

[6] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions[J], " *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[7] M. Mezmaz, N. Melab, Y. Kessaci, et al. "A parallel bi-objective hybrid metaheuristic for Energy-aware scheduling for cloud computing systems[J]," *J. Parallel Distrib. Comput.*, 2011, vol. 71, no. 11, pp. 1497–1508.

[8] N. Auluck, S. Betha, and B. Mangipudi, "Contention aware energy efficient scheduling on heterogeneous multiprocessors [J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1251–1264, May 1, 2015.

[9] B. Dorronsoro, S. Nesmachnow, J. Taheri, et al. "A hierarchical approach for Energy-efficient scheduling of large workloads in multicore distributed systems[J]," *Sustainable Comput.: Informat. Syst.*, vol. 4, no. 4, pp. 252–261, 2014.

[10] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times[J]," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 18–28, Jan. 2005.

[11] C. Hsu and W. Feng, "A power-aware run-time system for high-performance computing[C], in *Proc. ACM/IEEE Conf. Supercomput.*, 2005: 1.

[12] Y. Ma, B. Gong, R. Sugihara, and R. Gupta, "Energy-efficient deadline scheduling for heterogeneous systems," *J. Parallel Distrib. Comput*, vol. 72, no. 12, pp. 1725–1740, Dec. 2012.

[13] P. A. La Fratta and P. M. Kogge, "Energy-efficient multithreading for a hierarchical heterogeneous multicore through locality-cognizant thread generation[J]," *J. Parallel Distrib. Comput.*, vol. 73, no. 12, pp. 1551–1562, 2013.

[14] L. A. Barroso and U. Holzle, "The case for energy-proportional computing[J]," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[15] R. Raghavendra, P. Ranganathan, V. Talwar, et al., "No power struggles: Coordinated multi-level power management for the data center[C]," *ACM SIGARCH Comput. Archit. News.*, vol. 36, no. 1, pp. 48–59, 2008.

[16] S. Nesmachnow, B. Dorronsoro, J. E. Pecero, et al., "Energy-aware scheduling on multicore heterogeneous grid computing systems[J]," *J. Grid Comput.*, vol. 11, no. 4, pp. 653–680, 2013.

[17] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems[J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[18] L. Mashayekhy, M. M. Nejad, D. Grosu, et al., "Energy-aware scheduling of MapReduce jobs big data (bigdata congress)," *IEEE Int. Congress*, pp. 32–39, 2014.

[19] D. S. Johnson and M. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[20] A. Gandhi, M. Harchol-Balter, R. Das, et al.. "Optimal power allocation in server farms[C]," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 1, pp. 157–168, 2009.

[21] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing[J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[22] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems[J]," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533–544, Jun. 2003.

[23] D. Bozdag, F. Ozguner, and U. V. Catalyurek, “Compaction of schedules and a two-stage approach for duplication-based DAG scheduling[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 6, pp. 857–871, Jun. 2009.
[24] E. S. H. Hou, N. Ansari, and H. Ren, “A genetic algorithm for multiprocessor scheduling[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113–120, Feb. 1994.
[25] A. S. Wu, H. Yu, S. Jin, et al. “An incremental genetic algorithm approach to multiprocessor scheduling[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 824–834, Sep. 2004.
[26] A. Swiecicka, F. Seredynski, and A. Y. Zomaya, “Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 3, pp. 253–262, Mar. 2006.
[27] M. H. Kashani and M. Jahanshahi, “Using simulated annealing for task scheduling in distributed systems[C],” in *Proc. Int. Conf. Comput. Intell., Model. Simul.*, 2009, pp. 265–269.
[28] F. Ferrandi, P. L. Lanzi, C. Pilato, et al. “Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems[J], *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.
[29] D. H. Wolpert, and W. G. Macready, “No free lunch theorems for optimization[J],” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[30] P. Ross, “Hyper-heuristics,” in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA: Kluwer, 2005, pp. 529–556.
[31] K. H. Han and J. H. Kim, “Quantum-inspired evolutionary algorithm for a class of combinatorial optimization[J],” *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 580–593, Dec. 2002.
[32] G. C. Sih, and E. A. Lee, “A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures[J],” *Parallel and Distributed Systems, IEEE Transactions on*, 1993, vol. 4, no. 2, pp. 175–187.
[33] R. Bajaj and D. P. Agrawal, “Improving scheduling of tasks in a heterogeneous environment[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 2, pp. 107–118, Feb. 2004.
[34] D. Bozdag, F. Ozguner, and U. V. Catalyurek, “Compaction of schedules and a two-stage approach for duplication-based DAG scheduling[J],” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 6, pp. 857–871, Jun. 2009.
[35] Y. C. Lee and A. Y. Zomaya, “Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling[C],” in *Proc. 9th IEEE/ACM Int. Symp. Cluster Comput. Grid*, 2009, pp. 92–99.
[36] K. Li, “Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers[J],” *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668–1681, Dec. 2012.
[37] E. Burke, G. Kendall, J. Newall, et al. “Hyper-heuristics: An emerging direction in modern search technology[J],” in *Handbook of Meta-Heuristics*, International series in operations research and management science. Dordrecht, Netherlands: Springer, 2003, pp. 457–474.
[38] E. K. Burke, G. Kendall, and E. Soubeiga, “A Tabu-search hyperheuristic for timetabling and rostering[J],” *J. Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.
[39] E. K. Burke, S. Petrovic, and R. Qu, “Case-based heuristic selection for timetabling problems[J],” *J. Scheduling*, vol. 9, no. 2, pp. 115–132, 2006.
[40] K. A. Dowsland, E. Soubeiga, and E. Burke, “A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation[J],” *Eur. J. Oper. Res.*, vol. 179, no. 3, pp. 759–774, 2007.
[41] E. K. Burke, G. Kendall, M. Misir et al. “Monte carlo hyper-heuristics for examination timetabling[J],” *Ann. Operations Res.*, vol. 196, no. 1, pp. 73–90, 2012.
[42] J. Gascón-Moreno, S. Salcedo-Sanz, B. Saavedra-Moreno, et al. “An Evolutionary-based Hyper-heuristic approach for optimal construction of group method of data handling networks[J],” *Inf. Sci.*, vol. 247, pp. 94–108, 2013.
[43] E. K. Burke, M. Hyde, and G. Kendall, “Evolving bin packing heuristics with genetic programming,” in *Proc. 9th Int. Conf. Parallel Problem Solving Nature*, Sep. 9–13, 2006, pp. 860–869.
[44] A. S. Fukunaga, “Automated discovery of local search heuristics for satisfiability testing[J],” *Evol. Comput.*, vol. 16, no. 1, pp. 31–61, 2008.
[45] E K, M. Hyde, G. Kendall, et al. “A genetic programming hyperheuristic approach for evolving 2-D strip packing heuristics[J],” *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 942–958, Dec. 2010.
[46] K. Deb, “An efficient constraint handling method for genetic algorithms[J],” *Comput. Methods Appl. Mech. Eng.*, vol. 186, no. 2, pp. 311–338, 2000.
[47] F. Hoffmeister and J. Sprave, “Problem-independent handling of constraints by use of metric penalty functions[C],” in *Proc. 5th Conf. Evol. Program.*, 1996, pp. 289–294.
[48] T. Hey, “Quantum computing: An introduction[J],” *Comput. Control Eng. J.*, vol. 10, no. 3, pp. 105–112, 1999.
[49] Z. Li, G. Rudolph, and K. Li, “Convergence performance comparison of quantum-inspired multi-objective evolutionary algorithms [J],” *Comput. Math. Appl.*, vol. 57, no. 11, pp. 1843–1854, 2009.
[50] Z. Li, B. Xu, and L. Yang, et al. “Quantum evolutionary algorithm for Multi-robot coalition formation[C],” in *Proc. 1st ACM/SIGEVO Summit Genetic Evol. Comput.*, 2009, pp. 295–302.
[51] S. J. Kim and J. C. Browne, “A general approach to mapping of parallel computation upon multiprocessor architectures[C],” in *Proc. Int. Conf. Parallel Process.*, 1988, vol. 3, no. 1, p. 8.

Shaomiao Chen received the MSc degree in computer science and technology from Hunan University, Changsha, China, in 2014. He is currently working toward the PhD degree in the Hunan University. His research interests include parallel computing, evolutionary computation, and scheduling optimization.

Zhiyong Li received the MSc degree in system engineering from National University of Defense Technology, Changsha, China, in 1996 and the PhD degree in control theory and control engineering from Hunan University, Changsha, China, in 2004. Currently, he is a full professor with Hunan University, a member of China Computer Federation (CCF). His research interests include embedded computing system, visual object tracking, dynamic multiobjective evolutionary algorithm, and tasks scheduling optimization in cloud computing. He obtained several awards from academic organizations and conferences, such as the Champion of the Future Challenge: Intelligent Vehicles and Beyond, FC’09, which was hosted by the National Natural Science Fund Committee of China in 2009. He is a member of the IEEE.

Bo Yang received the MS degree in computer science from Hunan university,China, in 2005. He is currently working toward the PhD degree at Hunan university of China. His current research focus on resource management in cloud computing system, game theory, and multiobjective optimization.

Günter Rudolph received the diploma and doctoral degree in computer science from Dortmund University in 1991 and 1996, respectively. He has been with Informatics Center Dortmund (ICD), Collaborative Research Center on Computational Intelligence (SFB 531), and Parsytec AG, Aachen, Germany. Since 2005, he has been a full professor of computer science at TU Dortmund University. His research interests include parallel computing, multiobjective optimization, and digital entertainment technologies. He is a member of the IEEE.