Information Sciences 181 (2011) 567–581

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

# A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system

Yun Wen, Hua Xu *, Jiadong Yang

*State Key Laboratory on Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, PR China*

ARTICLE INFO

*Article history:*
Received 2 February 2010
Received in revised form 28 September 2010
Accepted 2 October 2010

*Keywords:*
Variable neighborhood search
Genetic algorithm
Hybrid metaheuristic
Memetic algorithm
Heterogeneous multiprocessor scheduling

ABSTRACT

Effective task scheduling, which is essential for achieving high performance in a heterogeneous multiprocessor system, remains a challenging problem despite extensive studies. In this article, a heuristic-based hybrid genetic-variable neighborhood search algorithm is proposed for the minimization of makespan in the heterogeneous multiprocessor scheduling problem. The proposed algorithm distinguishes itself from many existing genetic algorithm (GA) approaches in three aspects. First, it incorporates GA with the variable neighborhood search (VNS) algorithm, a local search metaheuristic, to exploit the intrinsic structure of the solutions for guiding the exploration process of GA. Second, two novel neighborhood structures are proposed, in which problem-specific knowledge concerned with load balancing and communication reduction is utilized respectively, to improve both the search quality and efficiency of VNS. Third, the proposed algorithm restricts the use of GA to evolve the task-processor mapping solutions, while taking advantage of an upward-ranking heuristic mostly used by traditional list scheduling approaches to determine the task sequence assignment in each processor. Empirical results on benchmark task graphs of several well-known parallel applications, which have been validated by the use of non-parametric statistical tests, show that the proposed algorithm significantly outperforms several related algorithms in terms of the schedule quality. Further experiments are carried out to reveal that the proposed algorithm is able to maintain high performance within a wide range of parameter settings.



## 1. Introduction

As a most promising approach to meet the rising computational requirements, parallel processing approach poses a number of problems not encountered in traditional sequential processing [9,23], the most important of which is the multiprocessor scheduling issue. In general, an originally large program can be decomposed into a set of smaller tasks prior to parallel processing. These smaller tasks almost always have dependencies representing the precedence constraints, in which the results of other tasks are required before a particular task can be executed. Hence, the goal of a task scheduling algorithm is typically to schedule all the tasks on the given number of available processors so as to minimize the overall length of time required to execute the entire program, namely makespan, without violating precedence constraints. Based on various characteristics of the decomposed tasks to be scheduled and the multiprocessor system, as well as the availability of *a priori* information regarding the processing time, the multiprocessor scheduling problem can be categorized into many different classes [12,29]. In this article, only the *static* scheduling problem is addressed, in which all information needed for

* Corresponding author.
*E-mail address:* xuhua@mail.tsinghua.edu.cn (H. Xu).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.10.001

scheduling, including task processing times, data dependencies, and communication costs between dependent tasks, are known before program execution.

Given the NP-complete complexity for searching an optimal solution in its general form [40], the multiprocessor scheduling problem remains an open field in spite of extensive studies. Traditional scheduling research focused on the heuristic-based algorithms, an important class of which is the so-called list scheduling algorithms [29,38]. The basic idea of list scheduling consists in maintaining an ordered list of tasks by assigning priority for each task according to some greedy heuristics. Tasks are then selected in the order of their priorities and the highest-priority ready task is removed from the list to be assigned to a processor which allows the earliest start time. Traditional list scheduling algorithms usually assume a homogeneous multiprocessor system in which all processors are of the same processing ability and fully connected [2,41], while recent studies have been diverted to task scheduling for heterogeneous multiprocessor systems where the execution time of a task may vary among different processors [38]. The heuristic-based scheduling algorithms are always efficient since they narrow the search down to a very small portion of the solution space by means of greedy heuristics. However, due to the greedy nature, heuristic-based approaches are not likely to produce consistent results on a wide range of problems, especially when the complexity of the scheduling problem increases.

In attempts to obtain schedules of better quality, many well-known metaheuristics, including genetic algorithm (GA) [1,6,8,21,33,37,42], artificial immune system (AIS) [44], ant colony optimization (ACO) [5], particle swarm optimization (PSO) [34], simulated annealing (SA) [26], tabu search (TS) [36], and variable neighborhood search (VNS) [32], have been adopted. Generally, metaheuristics can be divided into trajectory methods (also named local search heuristics) and population-based methods. Population-based methods deal with a set of solutions in every iteration of the algorithm while trajectory methods only deal with a single solution [3]. As one of the most studied population-based methods, GA shows robust performance with various scheduling problems, for it has a powerful global exploration ability of concurrently tracking a set of solutions. Plenty of empirical results demonstrate that GA-based methods always outperform traditional heuristic-based scheduling algorithms regarding the schedule quality [6,21,42]. However, GA usually takes more computing efforts to locate the optimal in the region of convergence [42], owing to the lack of local search ability. On the other hand, the trajectory method, such as VNS [10], has shown its potential in exploiting the promising regions in the search space with high quality solutions. Nevertheless, it is still prone to premature convergence traps due to the limited exploration ability. Thus, it's a natural choice to consider the hybridization of metaheuristics, also named memetic algorithm (MA) in some literatures [27], which has recently been applied to solve scheduling problems [4].

The approach proposed in this article is largely inspired by the fact that each type of scheduling technique has its own strengths and weaknesses while appearing that they are complementary to each other. This paper attempts to present a novel heuristic-based hybrid genetic-variable neighborhood search (GVNS) algorithm for solving the heterogeneous multiprocessor scheduling problem, which improves upon the standard GA in three aspects. First, the proposed algorithm incorporates GA and VNS to obtain a balance between the exploration and exploitation of the search space, which is crucial to the success of metaheuristics. Next, two common scheduling strategies, load balancing and communication reduction, are utilized in our proposed neighborhood structures in VNS, which further improves the local search efficiency. Finally, unlike many existing GA approaches which directly use GA to evolve the priorities of the tasks that in turn determine the final schedule solution, the proposed GVNS restricts the use of GA and VNS to find optimal task-processor mapping while leaving the task sequence assignment to an upward-ranking heuristic originally used in HEFT [38], a list scheduling algorithm.

In the next section, a formal statement of the studied heterogeneous multiprocessor scheduling problem is provided. Section 3 introduces some existing related researches on the multiprocessor scheduling problem. Section 4 presents the various features of the proposed GVNS. The benchmark problems, parameter settings, experimental results and analysis are explained in Section 5. Finally, conclusions and suggestions for future research are drawn in Section 6.

## 2. Problem formulation

The static multiprocessor scheduling problem is typically given by two inputs: a multiprocessor system in which tasks can be solved and a parallel program to be computed. Generally, the target multiprocessor system is composed of a network of processors, each of which has a local memory so that inter-processor communications rely solely on message-passing [29]. In this paper, the studied multiprocessor system model is assumed to be with the following characteristics: (1) heterogeneous; (2) non-preemptive; (3) fully connected network; (4) communication links with uniform bandwidth; (5) task duplication is NOT allowed; (6) each processor has an independent I/O unit that allows for communication and computation to be performed simultaneously.

Meanwhile, the parallel program is typically decomposed into smaller tasks with precedence constraints, which are described as a directed acyclic graph (DAG). In general, a DAG $G = (V, E)$ consists of a set $V$ of $v$ nodes and a set $E$ of $e$ edges. A node in the DAG represents a decomposed task, which must be executed sequentially without preemption in the same processor. Hereafter, the terms *node* and *task* will be used interchangeably. The computation cost of a particular task $n_i \in V$ on the processor $p_j$ is denoted as $w_{i,j}$. Each edge $e_{i,j} \in E$ represents precedence constraints between task $n_i$ and $n_j$, which means that the result of task $n_i$ has to be transmitted to task $n_j$ before task $n_j$ starts its execution. Each edge $e_{i,j} \in E$ is associated with a nonnegative weight $c_{i,j}$ representing the communication cost between the interdependent tasks $n_i$ and $n_j$. Note that the real communication cost is equal to zero when the interdependent pair of tasks are assigned to the same processor. The source

node of an edge is called the *predecessor* while the sink node is called the *successor*. The task with no predecessor is called the *entry* task while the task with no successor is called the *exit* task.

Assume that there is a task graph with $T$ tasks to be assigned to a multiprocessor system with $P$ processors. Given a partial schedule solution, considering scheduling the highest-priority ready task $n_i$ on the processor $p_j$, its earliest start time $T_{EST}(n_i, p_j)$ can be defined as

$$T_{EST}(n_i,\ p_j) = \max\{T_{avail}(p_j),\ T_{ready}(n_i, p_j)\}, \tag{1}$$

where $T_{avail}(p_j)$ is the time when processor $p_j$ is available to the execution of the task $n_i$. It can be defined as

$$T_{avail}(p_j) = \max_{n_k \in exe(p_j)} \{T_{AFT}(n_k)\}, \tag{2}$$

where $exe(p_j)$ denotes the set consisting of all the tasks that have already been scheduled on the processor $p_j$ while $T_{AFT}(n_k)$ represents the actual finish time when the task $n_k$ actually finishes its execution. Moreover, $T_{ready}(n_i, p_j)$ in Eq. (1) denotes the time when all the data needed for the execution of task $n_i$ have been transmitted to the processor $p_j$, which can be defined as

$$T_{ready}(n_i, p_j) = \max_{n_k \in pred(n_i)} \{T_{AFT}(n_k) + c_{k,i}\} \tag{3}$$

where $T_{AFT}(n_k)$ has the same meanings as in Eq. (2) and $pred(n_i)$ denotes the set consisting of all the immediate predecessors of the task $n_i$. Note that $c_{k,i}$ equals to 0 if the task $n_k$ has been scheduled to the same processor $p_j$.

Given that the task $n_i$ is allocated on the processor $p_j$ with non-preemptive processing approach, its earliest finish time $T_{EFT}(n_i, p_j)$ can be defined as

$$T_{EFT}(n_i, p_j) = T_{EST}(n_i, p_j) + w_{i,j}, \tag{4}$$

where $w_{i,j}$ represents the execution time of the task $n_i$ on the processor $p_j$.

After the task $n_i$ is explicitly scheduled on the processor $p_j$, the $T_{EST}(n_i, p_j)$ and $T_{EFT}(n_i, p_j)$ are assigned to $T_{AST}(n_i)$ and $T_{AFT}(n_i)$ respectively. The overall schedule length of the entire parallel program, namely *makespan*, is the largest finish time among all tasks, which is equivalent to the actual finish time of the exit node $n_{exit}$.

$$makespan = \max_{n_i \in V} \{T_{AFT}(n_i)\} = T_{AFT}(n_{exit}). \tag{5}$$

A simple nine-node task graph with communication cost of each edge is shown in Fig. 1(a). The computation cost matrix consisting of computation costs of each node on different processors is detailed in Fig. 1(b). One sample schedule of the task graph is illustrated in Fig. 1(c), which achieves a makespan of 106 by allocating four available processors.

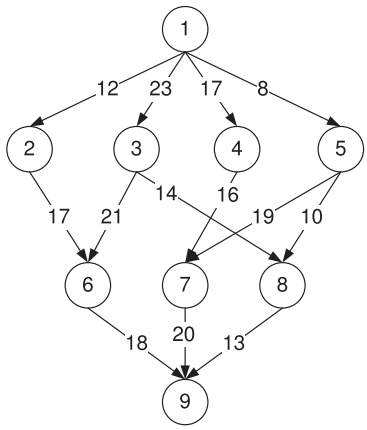
## 3. Related work

In general, the static multiprocessor scheduling problem is an NP-complete problem [40], except for three simplified cases [7,22,35]. Over the last decade, various scheduling approaches have been proposed, which can be classified into two main categories, *deterministic* and *non-deterministic*.

Deterministic algorithms are based on heuristics extracted from developers' intuitions, the most representative of which are the so-called list scheduling algorithms [2,24,28,29,38]. As to the heterogeneous multiprocessor scheduling problem, HEFT algorithm proposed by Topcuoglu et al. [38] uses an upward-ranking heuristic for the task priority calculation, in which the information of average execution cost of a task is utilized. It selects the task with the highest upward rank value at each step and assigns the selected task to the processor that minimizes the earliest finish time with an insertion-base approach. Simulation results indicate that HEFT outperforms the other list scheduling algorithms in terms of both schedule quality and computational cost.

In contrast to deterministic algorithms, non-deterministic algorithms incorporate metaheuristic in the search process, which show robust performance on a variety of optimization problems. Many famous metaheuristic, such as GA [19,30,39], AIS [31,44,46], ACO [5], PSO [34], SA [26], TS [36], and VNS [10] have been successfully applied to a broad range of scheduling problems. Nevertheless, to trade off better performance in terms of schedule quality, non-deterministic algorithms always have much higher computational cost.

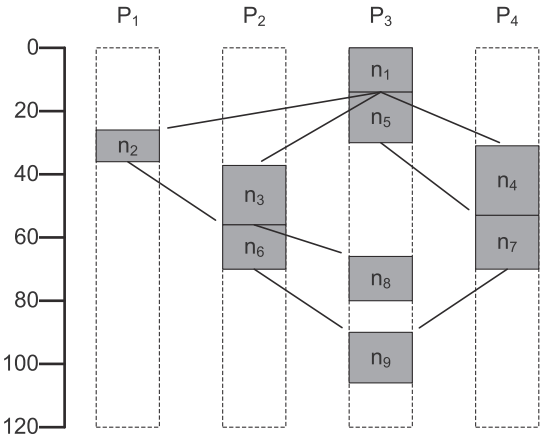
As a population-based method originally developed in [19], GA features itself by a chromosome encoding for solution representation and some genetic operators inspired by the nature selection mechanism, which has attracted plenty of attentions in the field of multiprocessor scheduling [6,8,21,42,45]. Wu et al. [42] develop an incremental GA (IGA) approach to multiprocessor scheduling, in which the population contains not only individuals representing feasible solutions but also those corresponding to infeasible ones. This approach eliminates the need for special genetic operators or repair mechanisms to ensure the validity of evolved individuals. Experimental results show that IGA has a robust performance for both homogeneous and heterogeneous multiprocessor scheduling problems. Bonyadi et al. [6] propose a bipartite GA (BGA), each part of which is based on different genetic schemes. The first part is used to find an adequate sequence of tasks while the second one is used to search for the best match processors. BGA is claimed to generate satisfactory results with at least 10% less iterations compared to IGA.




(a) Task graph with communication cost

| Task | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 1 | 14 | 16 | 14 | 17 |
| 2 | 10 | 12 | 12 | 14 |
| 3 | 17 | 19 | 15 | 16 |
| 4 | 20 | 22 | 18 | 22 |
| 5 | 21 | 18 | 16 | 20 |
| 6 | 12 | 14 | 14 | 16 |
| 7 | 16 | 16 | 20 | 17 |
| 8 | 19 | 17 | 20 | 18 |
| 9 | 14 | 15 | 16 | 16 |

(b) Computation cost matrix




(c) A sample schedule

**Fig. 1.** Illustration of task scheduling in heterogeneous multiprocessor system without task duplication.

On the other hand, VNS is a simple but effective trajectory metaheuristic firstly proposed in [32]. The principle of VNS involves using two or more neighborhood structures and systematically changing the neighborhood within a local search process. Unlike many other metaheuristics, the basic scheme of VNS and its extensions requires few and sometimes no parameters. This simplicity promotes the development of VNS in both its methods and applications [18]. Davidović et al. [10] firstly apply VNS heuristics to multiprocessor scheduling with communication delays. Their approach uses several neighborhood structures analogous to the well-known ones used in solving the travelling salesman problem (TSP). However, the totally random search strategies used in these neighborhood structures reduce the efficiency of VNS.

There also exist some reported works on the hybridization of both deterministic and non-deterministic algorithms for multiprocessor scheduling [16,44]. Yu in [44] proposes an AIS-based algorithm for task scheduling in heterogeneous multiprocessor systems, in which the scheduling process is separated into two phases, task-processor mapping and task sequence assignment. A heuristic similar to HEFT is adopted to take charge of the task sequence assignment. Empirical study on benchmark task graphs demonstrates that the AIS-based algorithm can deliver schedule solutions with better quality than HEFT.

## 4. The proposed algorithm

This section presents a detailed description of the proposed GVNS algorithm, including the whole algorithmic flow and various features borrowed from GA, VNS and list scheduling heuristics.

### *4.1. Framework*

The algorithmic flow of the proposed GVNS is depicted in Fig. 2. The optimization process begins with initializing the population of GA based on a randomized procedure. After the population initialization, every schedule solution represented by

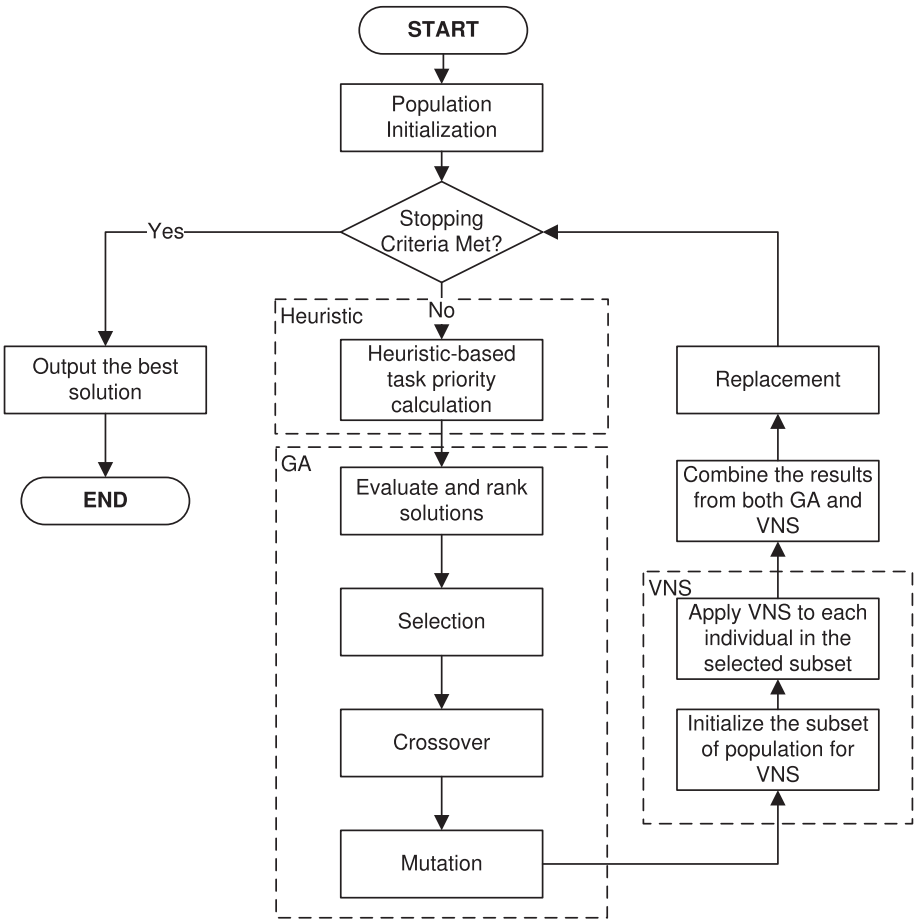



**Fig. 2.** Flowchart of the proposed GVNS.

each individual undergoes a heuristic-based task priority calculation (to be discussed in Section 4.3). Following the task prioritization, the population is evaluated and ranked with makespan being the fitness of each individual. A set of genetic operators, including binary tournament selection, random-one-point crossover, and flipping mutation, is then performed on the evaluated population. Thereafter, the population is sampled in order to select a subset consisting of individuals for VNS initialization. The local search procedure of VNS (to be detailed in Section 4.5) is then applied to each individual in the selected subset. Finally, the results from both GA and VNS are combined and some replacement strategies are introduced to update the population. The evolution process is repeated until the stopping criterion is satisfied.

### 4.2. Encoding of solutions

The population in GVNS consists of a group of individuals and each individual represents a candidate assignment solution to the given scheduling problem. Each solution is encoded as a string of integers while each integer in the string represents the index of the processor to which a task is assigned. Suppose that the multiprocessor scheduling problem consists of a multiprocessor system with $P$ available processor and a parallel program with $T$ tasks. Moreover, it is assumed that each task has a unique ID ranging from 1 to $T$ while each processor has a unique index ranging between 1 and $P$. The $i$th integer of each string represents the index of the processor to which the task $t_i$ is assigned. Fig. 3 shows the encoding of a solution corresponding to the schedule in Fig. 1(c).

This position-independent encoding mechanism implies the separation of task-processor mapping and task order assignment, which compose the two main phases of our proposed scheduling approach. The GA and VNS are used to evolve the

| 3 | 1 | 2 | 4 | 3 | 2 | 4 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|

**Fig. 3.** Encoded solution of the schedule given in Fig. 1(c).

task-processor mapping solutions, which ensure the exploration of search space. Meanwhile, a list scheduling heuristic is used to calculate task priorities in each processor, which guarantees the feasibility of every evolved solution. Unlike some existing GA-based approaches where invalid solutions may also be contained in the evolving population (e.g., IGA [42]), this encoding mechanism helps improve the efficiency of the proposed algorithm.

### 4.3. Heuristic-based task priority calculation

Given an encoded solution, in which each task is allocated to a distinct processor, the formation of the final schedule still requires the determination of the task order assignment on each processor. Then the overall schedule length is calculated to evaluate the fitness of the individual in GA. Intuitively, the priority of a given task should grow with its criticality to the potential reduction of makespan, while the calculation of makespan should take into account both the completion time of the task itself and completion time of the tasks that depend on it. The former can be easily calculated while the latter can not be accurately evaluated until the execution order of all the successor tasks is determined. Inspired by the HEFT in [38], an upward-ranking heuristic is applied to estimate the priority of each task. The upward rank of a task $t_i$, $rank(t_i)$ is defined recursively as follows:

$$rank(t_i) = \begin{cases} w_{i,p(t_i)}, & \text{if} \quad succ(t_i) = \emptyset, \\ w_{i,p(t_i)} + \max\limits_{t_j \in succ(t_i)} \left(rc_{i,j} + rank(t_j)\right), & \text{if} \quad succ(t_i) \neq \emptyset, \end{cases} \tag{6}$$

where $p(t_i)$ denotes the index of the processor to which the task $t_i$ is allocated, $w_{i,p(t_i)}$ denotes the computational cost of the task $t_i$ on the processor $p(t_i)$, $succ(t_i)$ denotes the set of immediate successors of task $t_i$, and $rc_{i,j}$ denotes the real communication cost between task $t_i$ and task $t_j$, which is further defined in Eq. (7):

$$rc_{i,j} = \begin{cases} c_{i,j}, & \text{if} \quad p(t_i) \neq p(t_j), \\ 0, & \text{if} \quad p(t_i) = p(t_j), \end{cases} \tag{7}$$

By means of the heuristic-based method above for task priority calculation, a task with the largest upward rank among all the ready tasks is always chosen to be performed first.

Although both use the notation of upward rank to determine the task priority, there are some subtle differences between the proposed approach and HEFT. First, as the encoded solution, in which each task has been assigned to a distinct processor, is given before the priority calculation, the proposed approach can take advantage of the execution time on the specified processor instead of the average among various processors. For the same reason, the proposed approach is able to give the real communication cost between dependent tasks while HEFT assumes that the communication cost always exists even though they may be assigned to the same processor. Hence, it is believed that our proposed approach will give a more accurate estimation of the task priorities.

### 4.4. Diversification using GA

The basic algorithm of the GA is the same as a classic GA [19]. Details specific to the proposed approach are described below.

#### 4.4.1. Population initialization

The initial population is initialized with randomly generated *population_size* individuals. Each individual consists of an encoded solution, in which each task is randomly assigned to a processor. In addition, it is assumed that the order of tasks obeys the topological sort of the task graph, which can be easily implemented by performing a topological sorting before task ID assignment.

#### 4.4.2. Fitness calculation and selection

Rather than the roulette wheel selection, which is usually incorporated with some scaling technique, the binary tournament selection, an ordinal selection scheme, is adopted in our approach. Thus, the makespan of a given solution can be directly used as its fitness. Since tasks have been prioritized by the upward-ranking heuristic, the procedure described in Section 2 can be directly applied to calculate the makespan of a given schedule solution.

#### 4.4.3. Crossover

Recall that each individual consists of a string of integers with each integer representing the processor to which the corresponding task is assigned. Crossover exchanges substrings between two individuals, which allows GA to explore new solutions while still retaining some structured parts of previously discovered solutions. Random one-point crossover operator is used in the proposed algorithm, in which a crossover point is randomly chosen for both parents. The segments to the right of the crossover point are exchanged to form two offspring. Fig. 4 exemplifies a random crossover operation. The crossover rate *crossover_rate* indicates the probability that a pair of parents will undergo crossover. Moreover, parents that do not crossover may still undergo mutation.

*4.4.4. Mutation*

The mutation rate *mutation_rate* gives the probability that an integer of the string will be changed, which means that the corresponding task is allocated to another processor. As a result, the expected number of mutations per individual is equal to the mutation rate multiplied by the string length of an individual. The mutation operator is crucial to preventing GA from premature convergence.

### 4.5. Intensification using VNS

Generally, GAs have been proven to be very good at shuffling the solution space but fail to intensify the search in promising regions. However, hybridization with local search methods may conquer this weakness and lead to powerful search schemes. Hence, VNS is adopted by our proposed approach to balance global exploration and local exploitation during the evolutionary process. The main steps of VNS in our proposed approach is described in Algorithm 1. As the essential factor of VNS, the two proposed neighborhood structures (i.e., line 2), which deal with both load balancing and communication reduction issues, will be detailed in Sections 4.5.2 and 4.5.3, respectively.

**Algorithm 1:** Algorithmic flow of VNS in the GVNS

1. Initialize the subset of the population *subpop* to undergo the local search process of VNS
2. Select the set of neighborhood structures $\mathcal{N}_k$ ($k = 1,2,3,\ldots,k_{\max}$)
3. **for** each individual $x$ in the *subpop* **do**
4. **repeat**
5. $k \leftarrow 1$
6. **while** $k \leqslant k_{\max}$ **do**
7. *Shaking*: Randomly generate a solution $x'$ from the $k$th neighborhood of $x$
8. *Local search*: Apply some local search method with $x'$ as the initial solution. The local optimum obtained is denoted as $x''$
9. *Move or not*: If $x''$ is better than the incumbent, move there $x \leftarrow x''$, and set $k \leftarrow 1$; otherwise set $k \leftarrow k + 1$
10. **end while**
11. **until** the termination condition is met
12. **end for**
13. Combine *subpop* with the offspring from GA

*4.5.1. Initialization*

Since the local search procedure is computationally intensive, the VNS algorithm is only applied to a subset of the population selected from the offspring, to which the crossover and mutation operators have been applied, using a random sampling technique. The sampling rate *sampling_rate* indicates the probability that an individual will be chosen to be a candidate for VNS.

*4.5.2. Neighborhood structure of load balance*

The effectiveness of VNS heavily depends on the switching of neighborhood structures in the shaking phase. Interestingly, it is observed that the best value for the parameter $k_{\max}$ is often two [17]. Rather than the simply randomized neighborhood structures used in [10], two novel targeted neighborhood structures are proposed in our VNS algorithm, in which two common scheduling strategies, load balancing and communication reduction, are utilized. Specifically, some problem-specific knowledge is introduced into these neighborhood structures for guiding the search process towards more promising regions of the search space as well as improving the searching efficiency. The neighborhood structure presented in this section is concerned with load balancing while the next section describes the one with communication reduction.

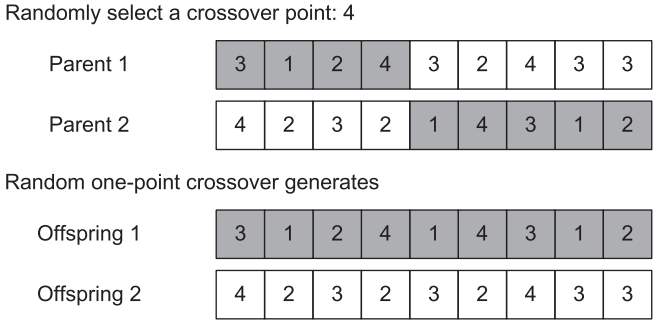



**Fig. 4.** Random one-point crossover operation illustration.

The *LoadBalance* neighborhood structure is designed to deal with the load balancing issue. The scheduling information used in this neighborhood structure is that which processor in a given schedule has the heaviest computation load. Let $T_{comp}(p_i)$ denote the computation load of processor $p_i$, then it can be defined as follows:

$$T_{comp}(p_i) = \sum_{n_k \in exe(p_i)} w_{k,i} \tag{8}$$

where $exe(p_i)$ denotes the set containing all the tasks scheduled to the processor $p_i$ and $w_{k,i}$ denotes the execution time of the task $n_k$ on the processor $p_i$.

Let $p_{comp}$ denote the processor with the largest $T_{comp}$. If there are two or more processors with the same $T_{comp}$, randomly choose one to break the tie. A node allocated in $p_{comp}$ is then randomly chosen to be transferred to another randomly selected processor. The computation process of *LoadBalance* neighborhood structure is depicted in Algorithm 2.

**Algorithm 2:** GenerateRandomNeighborhoodofLoadBalance ($s$)

1. **for** each processor $p_i$ in the solution $s$ **do**
2. $\quad T_{comp}(p_i) \leftarrow \sum_{n_k \in exe(p_i)} w_{k,i}$
3. **end for**
4. Choose the processor $p_{comp}$ with the largest $T_{comp}$
5. Randomly choose a node $n_{random}$ from $exe(p_{comp})$
6. Randomly choose a processor $p_{random}$ different from $p_{comp}$
7. Reallocate the node $n_{random}$ to the processor $p_{random}$
8. Encode and reschedule the changed solution $s'$
9. **return** $s'$

The instinct rationale behind *LoadBalance* neighborhood structure is that balancing load among various processors usually helps minimizing the makespan, especially when most tasks are allocated to only a few processors. Moreover, in the real-world distributed computing environment, unbalanced schedule always leads to insufficient utilization of the computing resources, which reduces the potential throughput.

*4.5.3. Neighborhood structure of communication reduction*

The *CommReduction* neighborhood structure concerns the communication reduction issue. The scheduling information used in this neighborhood structure is that which processor in a given schedule spends most time in inter-processor communications and idle waiting. Let $T_{comm}(p_i)$ denote the communication overhead of processor $p_i$, then it can be defined as follows:

$$T_{comm}(p_i) = T_{AFT}(p_i) - T_{comp}(p_i), \tag{9}$$

where $T_{comp}(p_i)$ is defined in Eq. (8) while $T_{AFT}(p_i)$ denotes the actual finish time of the processor $p_i$, which can be formally defined as follows:

$$T_{AFT}(p_i) = \max_{n_k \in exe(p_i)} T_{AFT}(n_k) \tag{10}$$

Let $p_{comm}$ denote the processor with the largest $T_{comm}$. If there are two or more processors with the same $T_{comm}$, use random choice to break the tie. Then a predecessor node of the nodes allocated in $p_{comm}$, which itself is not executed in $p_{comm}$, will be randomly chosen to be scheduled to $p_{comm}$. Algorithm 3 describes the computation process of *CommReduction* neighborhood structure.

**Algorithm 3:** GenerateRandomNeighborhoodofCommReduction ($s$)

1. **for** each processor $p_i$ in the solution $s$ **do**
2. $T_{comm}(P_i) \leftarrow T_{AFT}(P_i) - T_{comp}(P_i)$
3. **end for**
4. Choose the processor $p_{comm}$ with the largest $T_{comm}$
5. Set the candidate set empty, $cand \leftarrow \emptyset$
6. **for** each node $n_i$ in the set $exe(p_{comm})$ **do**
7. $\quad$ Calculate the set $pred(n_i)$ including all the predecessor nodes of $n_i$
8. $\quad$ Update $pred(n_i)$ with $pred(n_i) \leftarrow pred(n_i) - exe(p_{comm})$
9. $\quad cand \leftarrow cand + pred(n_i)$
10. **end for**
11. Randomly choose a node $n_{random}$ from $cand$
12. Reallocate the node $n_{random}$ to the processor $p_{comm}$
13. Encode and reschedule the changed solution $s'$
14. **return** $s'$

The underlying intuition of *CommReduction* neighborhood structure is that reducing communication overhead and idle waiting time of processors always results in a more effective schedule, especially given a relatively high unit communication cost. It is noticeable that in the shaking phase of the developed VNS, *LoadBalance* is always applied before *CommReduction*.

*4.5.4. Local search*

Since there are two variations of neighborhood structures used in our VNS, a specific local search is proposed for each neighborhood structure. Given a specific neighborhood structure, the local search can be simply summarized as finding the best neighborhood of the initial solution.

The reason why a thorough local search process is not performed can be explained as follows. Since VNS is hybridized with GA in our proposed algorithm, the local search process in VNS will proceed with the evolutionary process of GA. Seen from the entire evolutionary process, various steps, each of which is committed in the local search process of one generation, are merged to form a trajectory in the search space. Hence, from the view of efficiency, there is no need to perform a throughout search process in each generation, which may in turn result in a premature convergence.

*4.5.5. Termination condition*

Dual termination criteria are used in the VNS algorithm. The first one confines the upper bound of the local search iterations to 20 while the second one sets the maximum number of iterations without improvement to 3. Given either criterion is satisfied, the local search process of VNS will stop.

*4.6. Replacement strategy and stopping criterion*

After genetic operators and local search process are applied to generate the offspring, a replacement strategy is used to combining the current generation and offspring for reproduction. In the proposed GVNS, the current generation and offspring are merged and sorted in order of increasing makespan. Then the first *populations_size* solutions are selected to form the new generation.

The same stopping criterion as IGA in [42] is used in our proposed GVNS. A maximum number are chosen to be the upper bound of evolving generations.

## 5. Experimental details

In this section, the comparative evaluation of our proposed GVNS and several related algorithms given in Section 3 is presented. First, the benchmark task graphs used for performance evaluation are described. Next, the comparison metric and parameter settings used in the comparative study are given. Finally, the experimental results of various algorithms are presented, on which some non-parametric statistical tests are conducted.

*5.1. Test bed*

In order to test the performance of the proposed GVNS algorithm, task graphs that capture the representations and features of three well-known parallel applications, including the Gauss–Jordan elimination (GJ) [15], the Laplace equation solver (L) [41] and the fast Fourier transformation (FFT) [43], are treated as the benchmark problems. Rather than the number of tasks, a new parameter, matrix size $m$, is used to build the task graphs. Fig. 5(a)–(c) depict the task graphs of the Gauss–Jordan elimination, the Laplace equation solver, and the fast Fourier transformation having a metric size of 5, 3, and 4 respectively. As the graph structure remains the same with different matrix sizes, the total number of nodes in the task graph of Gaussian–Jordan elimination, Laplace equation solver, and fast Fourier transformation is equal to $(m^2+m)/2$, $m^2$, and $mlog_2m + 2m - 1$ respectively.

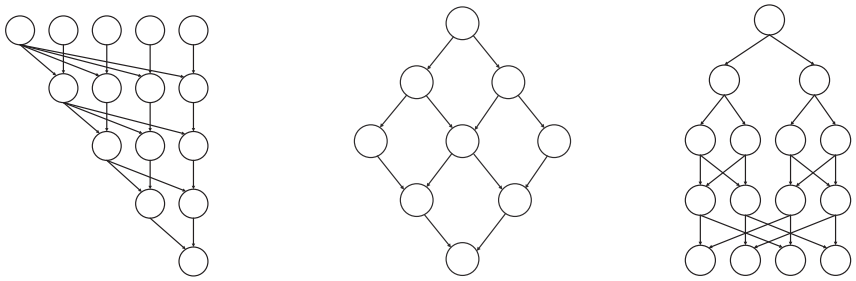



**Fig. 5.** Sample task graphs of three well-known applications.

**Table 1**
Parameter values for the benchmark problems.

| Parameter | Value |
|---|---|
| Matrix size of applications | {6,9,12} For Gauss–Jordan elimination<br>{4,6,8} For Laplace equation solver<br>{4,8,16} For Fast Fourier transformation |
| CCR value | {0.25,0.50,1.00,2.00} |
| The number of processors | {2,4,8} |

Each task graph is tested with different settings of communication to computation ratio (CCR) and processor number. Note that in our proposed approach the communication costs between different independent pairs of nodes are the same. For instance, if the CCR is 0.25 with the average computation cost set to be 40, the communication cost between two dependent tasks, if they are assigned to different processors, is 10. In addition, ten data configurations are randomly generated for each test case. The computation time of each task on each processor varies among the ten configurations while following Poisson distributions, just as those in AIS [44].

The parameters of our benchmark problems are summarized in Table 1. The benchmark task graphs with various metric sizes will be abbreviated as FFT-4, FFT-8, FFT-16, GJ-6, GJ-9, GJ-12, L-4, L-6, and L-8, respectively in the following sections.

### 5.2. Comparison metric and parameter setting

The performance comparisons of the algorithms are based on the metric, *speedup*, which is also used in [38,44]. The speedup value for a given schedule is computed by dividing the sequential execution time by the makespan of parallel execution. It can be defined as follows:

$$speedup = \frac{serial\_length}{makespan}, \tag{11}$$

where *serial_length* is computed by assigning all tasks to a single processor that minimizes the cumulative of the computation costs. It can be defined as follows:

$$serial\_length = \min_{p_j \in P} \left\{ \sum_{n_i \in V} w_{i,j} \right\} \tag{12}$$

where $w_{i,j}$ denotes the execution time of the task $n_i$ on the processor $p_j$.

In general, fine-tuning the parameters for every problem separately will give better outcomes. However, here we only aim to show the suitability of GVNS to solve the scheduling problems and the same parameter values are thus used to solve all problems. Besides, as we will see in Section 5.5, the proposed GVNS shows robust performance within a wide range of parameter settings. To sum up, the parameter settings used in the proposed GVNS and other algorithms in comparison are summarized in Table 2. Unless otherwise specified, the following values are used in our comparative evaluation.

### 5.3. Experimental results

The performance of the proposed algorithm is compared with those of the other four state-of-art scheduling algorithms, including HEFT in [38], VNS in [10], AIS in [44], and IGA in [42]. GVNS, HEFT, VNS and AIS prohibit task duplication while IGA allows task duplication. For each data configuration of each task graph, the average of the best speedups obtained over 50 runs is computed for GVNS, VNS, AIS, and IGA, while HEFT, a deterministic algorithm, is run only once. The results of ten data configurations are further averaged in each task graph. The experimental results of all the benchmarks are summarized in Table 3. The best result obtained in each test case is marked in bold font. In the IGA column of this table, the symbol – denotes that for the corresponding test cases, the IGA algorithm finds no valid solution after 3000 generations in all 50 runs,

**Table 2**
Parameter settings used in the proposed GVNS.

| Parameter | GVNS | AIS | IGA | VNS |
|---|---|---|---|---|
| Population size | 400 | 400 | 400 | – |
| Number of generations | 3000 | 3000 | 3000 | 3000 |
| Selection operator | Binary tournament | Roulette wheel | Binary tournament | – |
| Crossover operator | Random one-point | – | Random one-point | – |
| Mutation rate | 0.02 | 0.1 | 0.005 | – |
| Crossover rate | 0.8 | – | 0.8 | – |
| Sampling rate | 0.1 | 0.1 | – | – |

**Table 3**
Average speedups for the all the benchmark task graphs.

| Task | CCR | $p = 2$ | | | | | $p = 4$ | | | | | $p = 8$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | | GVNS | AIS | VNS | IGA | HEFT | GVNS | AIS | VNS | IGA | HEFT | GVNS | AIS | VNS | IGA | HEFT |
| GJ-6 | 0.25 | **1.95** | 1.93 | 1.89 | 1.77 | 1.84 | **2.99** | 2.72 | 2.70 | 2.65 | 2.66 | **3.37** | 2.96 | 2.86 | 3.30 | 2.74 |
| | 0.50 | **1.90** | 1.88 | 1.84 | 1.72 | 1.79 | **2.74** | 2.47 | 2.45 | 2.47 | 2.43 | **2.90** | 2.61 | 2.51 | 2.81 | 2.39 |
| | 1.00 | **1.81** | 1.79 | 1.66 | 1.65 | 1.51 | **2.13** | 2.01 | 1.95 | 2.05 | 1.89 | **2.14** | 2.03 | 1.94 | 2.11 | 1.85 |
| | 2.00 | **1.66** | 1.61 | 1.47 | 1.47 | 1.31 | **1.75** | 1.56 | 1.47 | 1.63 | 1.37 | **1.72** | 1.49 | 1.38 | 1.67 | 1.27 |
| GJ-9 | 0.25 | **2.10** | 2.01 | 1.96 | – | 1.90 | **3.52** | 3.02 | 3.08 | – | 3.14 | **4.65** | 3.70 | 3.86 | *2.90* | 4.01 |
| | 0.50 | **2.08** | 1.99 | 1.94 | – | 1.89 | **3.31** | 2.82 | 2.90 | – | 2.97 | **3.99** | 3.25 | 3.33 | *2.80* | 3.39 |
| | 1.00 | **2.03** | 1.93 | 1.87 | – | 1.79 | **2.80** | 2.44 | 2.51 | – | 2.56 | **3.01** | 2.60 | 2.60 | *2.16* | 2.60 |
| | 2.00 | **1.91** | 1.79 | 1.72 | – | 1.64 | **2.25** | 1.87 | 1.84 | – | 1.80 | **2.37** | 1.86 | 1.82 | *1.65* | 1.76 |
| GJ-12 | 0.25 | **2.15** | 2.01 | 1.97 | – | 1.93 | **3.95** | 3.27 | 3.42 | – | 3.57 | **5.75** | 4.28 | 4.71 | – | 5.14 |
| | 0.50 | **2.14** | 2.00 | 1.95 | – | 1.88 | **3.78** | 3.12 | 3.29 | – | 3.45 | **5.04** | 3.86 | 4.20 | – | 4.53 |
| | 1.00 | **2.11** | 1.97 | 1.91 | – | 1.84 | **3.44** | 2.82 | 2.96 | – | 3.11 | **3.92** | 3.20 | 3.35 | – | 3.50 |
| | 2.00 | **2.06** | 1.89 | 1.82 | – | 1.76 | **2.88** | 2.25 | 2.34 | – | 2.41 | **3.01** | 2.33 | 2.37 | – | 2.40 |
| L-4 | 0.25 | **1.81** | 1.76 | 1.64 | 1.62 | 1.51 | **2.15** | 2.04 | 1.98 | 2.05 | 1.91 | **2.14** | 1.99 | 1.92 | 2.16 | 1.84 |
| | 0.50 | **1.68** | 1.68 | 1.61 | 1.57 | 1.52 | **1.88** | 1.76 | 1.74 | 1.85 | 1.71 | **1.88** | 1.70 | 1.68 | 1.88 | 1.64 |
| | 1.00 | **1.52** | 1.52 | 1.38 | 1.38 | 1.24 | **1.59** | 1.44 | 1.39 | 1.54 | 1.34 | **1.58** | 1.35 | 1.33 | 1.57 | 1.30 |
| | 2.00 | **1.24** | 1.23 | 1.20 | 1.24 | 1.16 | **1.27** | 1.12 | 1.10 | 1.26 | 1.07 | **1.26** | 1.00 | 1.06 | 1.26 | 1.10 |
| L-6 | 0.25 | **1.95** | 1.88 | 1.84 | – | 1.79 | **2.84** | 2.43 | 2.50 | – | 2.56 | **3.14** | 2.63 | 2.63 | *2.81* | 2.64 |
| | 0.50 | **1.91** | 1.83 | 1.78 | – | 1.72 | **2.54** | 2.18 | 2.24 | – | 2.29 | **2.65** | 2.24 | 2.29 | *2.21* | 2.34 |
| | 1.00 | **1.80** | 1.69 | 1.64 | – | 1.57 | **2.07** | 1.77 | 1.81 | – | 1.82 | **2.10** | 1.74 | 1.78 | *1.82* | 1.82 |
| | 2.00 | **1.58** | 1.44 | 1.40 | – | 1.37 | **1.64** | 1.31 | 1.37 | – | 1.42 | **1.61** | 1.23 | 1.33 | *1.27* | 1.41 |
| L-8 | 0.25 | **2.03** | 1.90 | 1.89 | – | 1.86 | **3.29** | 2.68 | 2.83 | – | 2.96 | **4.01** | 3.11 | 3.18 | – | 3.22 |
| | 0.50 | **1.99** | 1.86 | 1.83 | – | 1.80 | **3.00** | 2.45 | 2.60 | – | 2.74 | **3.37** | 2.69 | 2.74 | – | 2.77 |
| | 1.00 | **1.93** | 1.76 | 1.73 | – | 1.69 | **2.48** | 2.07 | 2.17 | – | 2.25 | **2.60** | 2.12 | 2.13 | – | 2.15 |
| | 2.00 | **1.79** | 1.55 | 1.54 | – | 1.53 | **2.00** | 1.53 | 1.67 | – | 1.80 | **1.99** | 1.49 | 1.58 | – | 1.66 |
| FFT-4 | 0.25 | **1.97** | 1.92 | 1.84 | 1.79 | 1.75 | **2.80** | 2.55 | 2.46 | 2.48 | 2.37 | **2.89** | 2.61 | 2.42 | 2.84 | 2.23 |
| | 0.50 | **1.92** | 1.88 | 1.79 | 1.69 | 1.69 | **2.35** | 2.22 | 2.15 | 2.28 | 2.05 | 2.45 | 2.24 | 2.09 | **2.53** | 1.92 |
| | 1.00 | **1.72** | 1.70 | 1.64 | **1.72** | 1.57 | 1.82 | 1.77 | 1.70 | **2.02** | 1.61 | 1.84 | 1.74 | 1.67 | **2.17** | 1.59 |
| | 2.00 | 1.46 | 1.40 | 1.25 | **1.54** | 1.10 | 1.50 | 1.39 | 1.32 | **1.74** | 1.23 | 1.49 | 1.30 | 1.27 | **1.77** | 1.22 |
| FFT-8 | 0.25 | **2.03** | 1.99 | 1.92 | – | 1.85 | **3.56** | 3.10 | 3.04 | *2.81* | 2.97 | **4.74** | 3.78 | 3.87 | *4.04* | 3.94 |
| | 0.50 | **2.01** | 1.96 | 1.90 | – | 1.82 | **3.35** | 2.91 | 2.92 | – | 2.92 | **4.07** | 3.36 | 3.29 | *3.62* | 3.22 |
| | 1.00 | **1.96** | 1.90 | 1.82 | – | 1.73 | **2.94** | 2.56 | 2.56 | *1.91* | 2.54 | **3.16** | 2.75 | 2.73 | *3.08* | 2.71 |
| | 2.00 | **1.86** | 1.80 | 1.74 | – | 1.67 | **2.46** | 2.02 | 2.01 | *2.01* | 1.99 | **2.48** | 2.02 | 2.03 | *2.37* | 2.04 |
| FFT-16 | 0.25 | **2.16** | 2.02 | 1.99 | – | 1.95 | **4.12** | 3.54 | 3.58 | – | 3.61 | **6.88** | 5.27 | 5.74 | – | 6.18 |
| | 0.50 | **2.15** | 2.01 | 1.98 | – | 1.94 | **4.09** | 3.44 | 3.46 | – | 3.47 | **6.40** | 4.90 | 5.38 | – | 5.84 |
| | 1.00 | **2.13** | 1.99 | 1.96 | – | 1.92 | **3.90** | 3.26 | 3.31 | – | 3.35 | **5.70** | 4.28 | 4.68 | – | 5.06 |
| | 2.00 | **2.09** | 1.95 | 1.91 | – | 1.88 | **3.51** | 2.88 | 3.03 | – | 3.16 | **4.37** | 3.32 | 3.63 | – | 3.94 |

while the *italic* numbers imply that for these corresponding test cases not every run of IGA is able to find a valid solution and the average values are calculated only from those runs that do find valid solutions.

It is obvious that our proposed GVNS consistently outperforms HEFT,VNS, and AIS in all the test cases. With the scale of the scheduling problem increasing (e.g. larger number of processors or tasks), the gap becomes more remarkable. Taking GJ-21 with CCR set to 0.25 and processor number set to 2 as an example, GVNS achieves an average speedup of 1.95, which is only 6% better than the speedup of 1.84 generated by HEFT. When the number of processors rises to 8, the average speedup obtained by GVNS becomes 3.37, which improves upon the 2.74 of HEFT by almost 23%. This trend implies that the proposed GVNS has an increased ability to explore a larger solution space.

On the other hand, it is notable that GVNS is able to deliver better schedule solutions than IGA in all the test cases except for those of FFT application with a large CCR value (e.g. ratio of 2.0) or a large number of processors (e.g. 8 processors). This result is consistent with several existing works [2,42], in which task duplication heuristics are described to have particular advantage over other methods with higher communication costs. Besides, a larger number of processors provide more computing resources for taking advantage of the task duplication heuristic to further reduce the potential communication overheads. However, IGA has its own weaknesses due to a lack of effective search ability given a larger solution space. When the number of tasks exceeds 36 (e.g. task graph of L-6), IGA rarely finds a valid schedule solution after 3000 generations.

### 5.4. Significance tests

In order to find significant differences among the results obtained by the proposed GVNS and the other three related algorithms, including AIS, VNS, and HEFT (since IGA is not able to deliver valid solutions in every test case, it is excluded from the investigation), statistical analysis is necessary. Since the obtained results may present neither normal distribution nor homogeneity of variance, we consider the use of non-parametric tests according to the recommendations made in [11,14]. Specifically, the Friedman test [13], as well as the Iman–Davenport test [25], is employed to check whether there are significant differences in the scheduling performance among the four algorithms. If statistically significant differences exist, then we can proceed with the Holm method [20] as a post hoc test, in which our proposed GVNS (used as the control method) is compared against the remaining ones. A level of significance $\alpha = 0.05$ is used in all the statistical tests.

We first compute the average rank of each algorithm in all the 108 test cases, the results of which are shown in Table 4. Then the Friedman test and the Iman–Davenport test are employed, which both reject the null hypothesis of equivalent performance. Table 5 presents the results, which validate the existence of significant differences among the performance of all the scheduling approaches.

Further, the Holm's method is performed as a post hoc test to calculate the adjusted *p*-values (APVs), which takes into account that multiple tests are conducted. The results are shown in Table 6. Since Holm's procedure rejects all hypotheses, it could be safely concluded that the control algorithm (the proposed GVNS) is statistically better regarding scheduling quality than the remaining methods, with a significant level of 0.05.

### 5.5. Parameter sensitivity analysis

Given a fixed population size, the performance of GVNS mainly depends on the parameter settings of mutation rate, crossover rate, and sampling rate. A throughout sensitivity analysis of these three parameters is performed to find that

**Table 4**
Average speedup Friedman rankings.

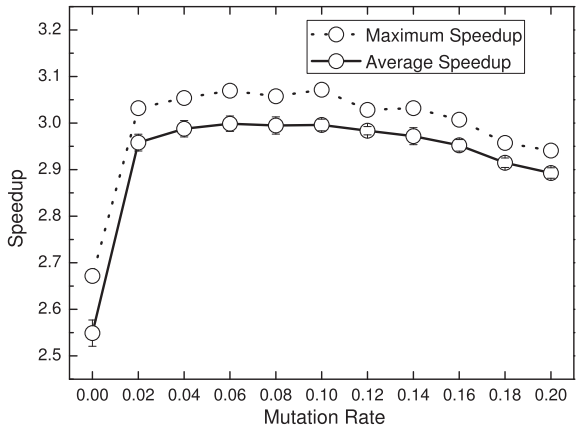
| Algorithm | GVNS | AIS | VNS | HEFT |
|---|---|---|---|---|
| Ranking | 1.0185 | 2.7546 | 2.9861 | 3.2407 |

**Table 5**
Results of Friedman's and Iman–Davenport's tests, $\alpha = 0.05$.

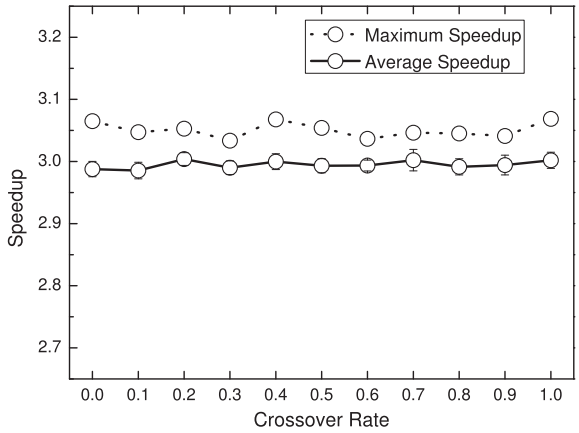
| Method | Statistic value | *p*-Value | Hypothesis |
|---|---|---|---|
| Friedman | 197.2917 | 1.0447E−10 | Rejected |
| Iman–Davenport | 166.6047 | 2.2204E−16 | Rejected |

**Table 6**
Results of Holm's method (GVNS is the control algorithm).

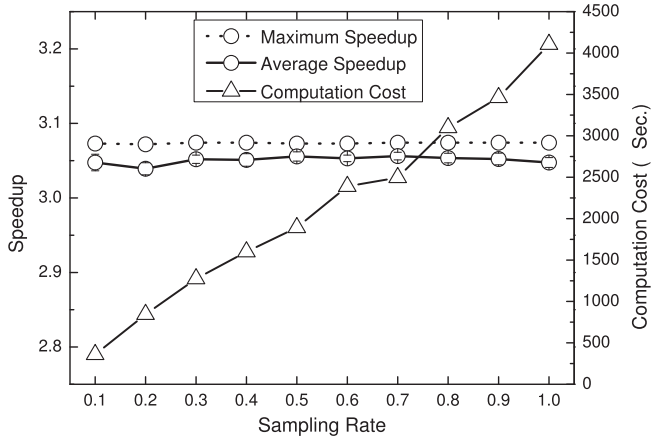
| *i* | Algorithm | *z* | *p*-Value | $\alpha/i$ | APV | Hypothesis |
|---|---|---|---|---|---|---|
| 3 | HEFT | 12.6491 | 1.1315E−36 | 0.0167 | 3.3945E−36 | Rejected |
| 2 | VNS | 11.1997 | 4.0896E−29 | 0.025 | 8.1792E−29 | Rejected |
| 1 | AIS | 9.8821 | 4.9770E−23 | 0.05 | 4.9770E−23 | Rejected |




(a) Mutation rate varying from 0 to 0.2.




(b) Crossover rate varying from 0 to 1.0.




(c) Sampling rate varying from 0.1 to 1.0.

**Fig. 6.** Results of parameter sensitivity analysis in the proposed GVNS.

our proposed GVNS is able to maintain robust performance within a wide range of parameter settings. Both the average speedup with 95% CI and the maximum speedup achieved in 50 runs over all the test cases are computed. Due to space limitations, only the results of the GJ-9 task graph are reported, with CCR set to 1.0 and processor number set to 8. Since mutation rate and crossover rate mainly impact the performance of GA, the hybridized VNS is disabled during the investigation of the two parameters.

The impact of mutation rate is investigated firstly, with crossover rate fixed to 1.0. Fig. 6(a) depicts the experimental results, which reveal that the algorithm performance shows slight difference with mutation rate varying from 0.02 to 0.2, except for the frontier point 0.

Thereafter, the sensitivity analysis of crossover rate is performed with mutation rate set to 0.1. The simulation results depicted in Fig. 6(b) indicate that the crossover rate has little impact on the search quality.

Finally, the sampling rate used in the initialization of VNS is examined with mutation rate set to 0.1 and crossover rate set to 1.0. The simulation results is illustrated in Fig. 6(c). It appears that the algorithm performance fluctuates very slightly while the computational cost of the algorithm increases sharply with the increasing sampling rates. Hence, a sampling rate as low as 0.1 is sufficient to guarantee search quality while maintaining high search efficiency.

## 6. Conclusion and future study

In this paper, a novel hybrid metaheuristic-based approach for solving task scheduling problem in the heterogeneous multiprocessor system has been proposed by means of incorporating GA with both VNS and an heuristic extracted from traditional list scheduling algorithms. Generally, our approach expands upon a traditional GA in two aspects. First, by means of hybridizing the global searching ability of the GA with the local improvement ability of the VNS, the balance between the exploration and exploitation of search space is enhanced. Second, problem-specific scheduling information is introduced to the neighborhood structures in the VNS for further improving the algorithm efficiency by guiding the local search towards a more promising region in the search space. In addition, task scheduling in our approach is separated into two phases of task-processor mapping and task priority determination. Our algorithm restricts the use of the GA and the VNS to evolve task-processor mapping solutions while taking advantage of an upward-ranking heuristic which improves upon HEFT for task priority calculation.

The performance of our proposed approach is compared with four related algorithms, HEFT, AIS, VNS and IGA, on the benchmark problems extracted from three well-known parallel applications. The experimental results show that our proposed algorithm consistently outperforms the other four algorithms in terms of schedule quality, with the exception that the task duplication-based IGA algorithm achieves the highest quality in several test cases with a high CCR value. Some non-parametric statistical tests for multiple comparison, including Friedman test, Iman–Davenport test and Holm's method as post hoc procedure, are conducted to validate the statistical significance of the performance difference. Finally, a comprehensive parameter sensitivity analysis is carried out to reveal that the proposed GVNS is able to maintain a high performance within a wide range of parameter settings. In addition, it appears that a partial sampling of the population (as low as 10%) used for VNS initialization is sufficient to guarantee the search quality.

In future research, the performance of the proposed approach can be tested using the data obtained from real distributed computing environment. Incorporating task duplication heuristic to further improve the schedule quality, especially given a high CCR value or a fine-grained task graph, is also considered, as well as parallelizing our proposed approach to improve the algorithm efficiency.

## Acknowledgments

The authors thank the helpful comments and suggestions from the editors and the anonymous reviewers, which have considerably enhanced the paper. This work is supported by National Natural Science Foundation of China (Grant No. 60875073), National Key Technology R&D Program of China (Grant No. 2009BAG12A08), and National S&T Major Project of China (Grant No. 2009ZX02001).

## References

[1] I. Ahmad, M. Dhodhi, Multiprocessor scheduling in a genetic paradigm, Parallel Computing 22 (1996) 395–406.
[2] I. Ahmad, Y. Kwok, On exploiting task duplication in parallel program scheduling, IEEE Transactions on Parallel and Distributed Systems 9 (1998) 872–892.
[3] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, John Wiley and Sons, Inc., Hoboken, New Jersey, 2005.
[4] J. Behnamian, M. Zandieh, S. Ghomi, Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm, Expert Systems with Applications 36 (2009) 9637–9644.
[5] F.F. Boctor, J. Renaud, A. Ruiz, S. Tremblay, Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems, in: Proceedings of 2008 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, Samos, Greece, 2008, pp. 142–149.
[6] M. Bonyadi, M. Moghaddam, A bipartite genetic algorithm for multi-processor task scheduling, International Journal of Parallel Programming 37 (2009) 462–487.
[7] E. Coffman, R. Graham, Optimal scheduling for two-processor systems, Acta Informatica 1 (1972) 200–213.
[8] R. Correa, A. Ferreira, P. Rebreyend, Scheduling multiprocessor tasks with genetic algorithms, IEEE Transactions on Parallel and Distributed Systems 10 (1999) 825–837.
[9] D. Culler, J. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publisher, San Francisco, 1998.
[10] T. Davidović, P. Hansen, N. Mladenović, Permutation-based genetic, tabu, and variable neighborhood search heuristics for multiprocessor scheduling with communication delays, Asia–Pacific Journal of Operational Research 22 (2005) 297–326.
[11] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research (7) (2006) 1–30.
[12] H. El-Rewini, T. Lewis, H. Ali, Task Scheduling in Parallel and Distributed Systems, Prentice Hall, 1994.
[13] M. Friedman, A comparison of alternative tests of significance for the problem of *m* rankings, Annals of Mathematical Statistics (11) (1940) 86–92.

[14] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064.
[15] A. Gerasoulis, T. Yang, Performance bounds for column-block partitioning of parallel Gaussian-elimination and Gauss–Jordan methods, Applied Numerical Mathematics 16 (1994) 283–297.
[16] C.K. Goh, E.J. Teoh, K.C. Tan, A hybrid evolutionary approach for heterogeneous multiprocessor scheduling, Soft Computing 13 (8–9, Special Issue) (2009) 833–846.
[17] P. Hansen, N. Mladenović, Variable neighborhood search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, 2003, pp. 145–184.
[18] P. Hansen, N. Mladenovic, J.M. Perez, Variable neighbourhood search: methods and applications, 4OR-A Quarterly Journal of Operations Research 6 (2008) 319–360.
[19] J. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, MA, 1992.
[20] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics (6) (1979) 65–70.
[21] E.S. Hou, N. Ansari, H. Ren, A genetic algorithm for multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 5 (2) (1994) 113–120.
[22] T. Hu, Parallel sequencing and assembly line problems, Operations Research 9 (1961) 841–848.
[23] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, Inc., New York, NY, 1993.
[24] J. Hwang, Y. Chow, F. Anger, C. Lee, Scheduling precedence graphs in systems with interprocessor communication times, SIAM Journal on Computing 18 (1989) 244–257.
[25] R. Iman, J. Davenport, Approximations of the critical region of the friedman statistic, Communications in Statistics (9) (1980) 571–595.
[26] A. Kalashnikov, V. Kostenko, A parallel algorithm of simulated annealing for multiprocessor scheduling, International Journal of Computer and Systems Sciences 47 (2008) 455–463.
[27] N. Krasnogor, J. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issues, IEEE Transactions on Evolutionary Computation 9 (2005) 474–488.
[28] B. Kruatrachue, T. Lewis, Duplication scheduling heuristics (DSH): a new precedence task scheduler for parallel processor systems, Technical Report, OR 97331, Oregon State University, Corvallis, 1987.
[29] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys 31 (1999) 406–471.
[30] Y. Li, Y. Yang, L. Zhou, R. Zhu, Observations on using problem-specific genetic algorithm for multiprocessor real-time task scheduling, International Journal of Innovative Computing, Information and Control 5 (9) (2009) 2531–2540.
[31] G.-C. Luh, C.-H. Chueh, A multi-modal immune algorithm for the job–shop scheduling problem, Information Sciences 179 (2009) 1516–1532.
[32] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (1997) 1097–1100.
[33] F. Omara, M. Arafa, Genetic algorithms for task scheduling problem, Journal of Parallel and Distributed Computing 7 (2010) 13–22.
[34] A. Omidi, A. Rahmani, Multiprocessor independent tasks scheduling using a novel heuristic PSO algorithm, in: Proceedings of the Second IEEE International Conference on Computer Science and Information Technology, Beijing, China, 2009, pp. 369–373.
[35] C. Papadimitriou, M. Yannakakis, Scheduling interval-ordered tasks, SIAM Journal on Computing 8 (1979) 405–409.
[36] S. Porto, C. Ribeiro, A tabu search approach to task scheduling on heterogeneous processors under precedence constraints, International Journal of High Speed Computing 7 (1995) 45–72.
[37] A. Singh, M. Sevaux, A. Rossi, A hybrid grouping genetic algorithm for multiprocessor scheduling, in: Proceedings of the Second International Conference on Contemporary Computing, Nioda, India, 2009, pp. 1–7.
[38] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (2002) 260–274.
[39] I.H. Toroslu, Y. Arslanoglu, Genetic algorithm for the personnel assignment problem with multiple objectives, Information Sciences 177 (2007) 787–803.
[40] J. Ullman, NP-complete scheduling problems, Journal of Computer and System Sciences 10 (1975) 384–393.
[41] M. Wu, D. Gajski, Hypertool: a programming aid for message-passing systems, IEEE Transactions on Parallel and Distributed Systems 1 (1990) 330–343.
[42] A. Wu, H. Yu, S. Jin, K. Lin, G. Schiavone, An incremental genetic algorithm approach to multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 15 (2004) 824–834.
[43] C. Yeh-Ching, S. Ranka, Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, in: Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, Minneapolis, Minnesota, USA, 1992, pp. 512–521.
[44] H. Yu, Optimizing task schedules using an artificial immune system approach, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, Atlanta, GA, USA, 2008, pp. 151–158.
[45] A. Zomaya, C. Ward, B. Macey, Genetic scheduling for parallel processor systems: comparative studies and performance issues, IEEE Transactions on Parallel and Distributed Systems 10 (1999) 795–812.
[46] X. Zuo, H. Mo, J. Wu, A robust scheduling method based on a multi-objective immune algorithm, Information Sciences 179 (2009) 3359–3369.