Egyptian Informatics Journal (2015) **16**, 199–212

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Multiobjective Variable Neighborhood Search algorithm for scheduling independent jobs on computational grid

S. Selvi [a,*], **D. Manimegalai** [b,1]

[a] *Department of Electronics and Communication Engineering, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur 628215, Tamil Nadu, India*
[b] *Department of Information Technology, National Engineering College, Kovilpatti 628503, Tamil Nadu, India*

Received 9 April 2014; revised 25 April 2015; accepted 1 June 2015
Available online 25 June 2015

KEYWORDS

Grid computing;
Flowtime;
Job scheduling;
Makespan;
Variable Neighborhood Search

Abstract Grid computing solves high performance and high-throughput computing problems through sharing resources ranging from personal computers to super computers distributed around the world. As the grid environments facilitate distributed computation, the scheduling of grid jobs has become an important issue. In this paper, an investigation on implementing Multiobjective Variable Neighborhood Search (MVNS) algorithm for scheduling independent jobs on computational grid is carried out. The performance of the proposed algorithm has been evaluated with Min–Min algorithm, Simulated Annealing (SA) and Greedy Randomized Adaptive Search Procedure (GRASP) algorithm. Simulation results show that MVNS algorithm generally performs better than other metaheuristics methods.

1. Introduction

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data and storage or network resources across dynamic and geographically dispersed organization [1]. Users can share grid resources by submitting computing tasks to grid system. Resources can be computers, storage space, instruments, software applications, and data, all connected through the Internet and a middleware layer that provides basic services for security, monitoring, resource management and so forth.

One of the main motivations of the grid computing paradigm has been the computational need for solving many complex problems from science, engineering, and business such as

* Corresponding author. Tel.: +91 8903484336; fax: +91 4639 243188.
E-mail addresses: mathini31@yahoo.co.in (S. Selvi), megalai_nec@yahoo.co.in (D. Manimegalai).
[1] Tel.: +91 9442636698; fax: +91 4632 232749.

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2015.06.001

hard combinatorial optimization problems, protein folding and financial modelling [2–4]. As a cooperative environment of solving problem, it is necessary for the grids to develop efficient job scheduling schemes and resource management policies in regard to their objectives, scope, and structure. However, there exists different and somewhat conflicting QOS objectives for management and security policies among the hierarchy based grid entities such as grid users (applications), grid resource administrative and virtual organization administrative. To increase the level of satisfaction of various grid entities, grid resource management system must use the scheduling strategy, which provides a compromise solution by considering several conflicting objectives.

Minimization of makespan is the most popular and extensively studied system-related optimization criterion. Makespan is an indicator of the general productivity of the grid system: Small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources. Considering makespan as a standalone criterion not necessarily implies the optimization of other objectives. Hence, it is necessary to devise the task scheduling algorithms in order to optimize both system-related and user-related objectives. One of the user-related objectives is the flowtime, which refers to the response time to the user submissions of task executions. Minimizing the value of flowtime means that the average response time of the grid system is being reduced. However, as discussed in [5], minimizing the makespan requires the most demanding jobs to be assigned to the fastest resource, at the expense of increasing the finish time of other jobs, and hence increasing flowtime. On the other hand, optimizing flowtime requires all jobs to finish quickly on the average, at the expense of having the most demanding jobs taking a longer completion time, thus increasing makespan. This justifies the search for algorithms that minimize both makespan and flowtime. Scheduling n jobs to m resources had been shown to be NP-complete [6]. Meta-heuristic approaches have shown their effectiveness for a wide variety of hard combinatorial problems and also for multi-objective optimization problems.

The main contribution of this work is the thorough experimental exploration of multiobjective VNS, with the problem specific neighborhood structures to solve the grid job scheduling problem, by minimizing the makespan and flowtime objectives. Efficient numerical results are reported in the experimental analysis performed on a set of 72 well known and large heterogeneous computing scheduling problem instances. The comparative study shows that the proposed MVNS is able to achieve high problem efficiency and outperforming the results of Min–Min algorithm, SA and GRASP algorithms.

Variable Neighborhood Search is a simple and effective meta-heuristic method developed to efficiently deal with the hard optimization problem. VNS is a framework for building heuristics, based upon systematic changes of neighborhoods both in descent phase, to find a local minimum, and in perturbation phase to emerge from the corresponding valley. VNS has also demonstrated good performance on industrial applications such as the design of an offshore pipeline network [7] and the pooling problem [8]. It has also been applied to real-world optimization problems, including optimization of a power plant cable layout [9], optical routing [10] and online resources allocation problem for ATM networks [11]. Applications of VNS are diverse which include the areas such as location problems, data mining, graph problems, mixed integer problems, scheduling problems, vehicle routing problems and problems in biosciences and chemistry [12].

2. Related works

Due to the popularization of distributed computing and the growing use of heterogeneous clusters in the 1990s [13,14], the heterogeneous computing scheduling problem (HCSP) became especially important. Hence many researchers paid attention in solving the HCSP. But the multiobjective HCSP variants that propose the simultaneous optimization of several efficiency metrics have been scarcely studied. Krauter et al. [15] provided a useful survey on grid resource management systems, in which most of the grid schedulers such as AppLes, Condor, Globus, Legion, Netsolve, Ninf and Nimrod use simple batch scheduling heuristics. Braun et al. [16] studied the comparison of the performance of batch queuing heuristics, Tabu Search (TS), GA and Simulated Annealing (SA) to minimize the *Makespan*. The results revealed that GA achieved the best results compared with the batch queuing heuristics. Some of the job scheduling algorithms are nature-inspired, e.g., SA [17], Ant Colony Optimization [18], Particle Swarm Optimization [19], Differential Evolution (DE) [20], parallel Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation (pCHC) [21]. There are also non-nature-inspired metaheuristics, such as TS [22], Threshold Accepting (TA) [23], and VNS algorithm [24]. Xhafa [25] studied the performance of Memetic algorithm (MA) with different local search algorithms including TS and VNS. The experimental results revealed that MA + TS hybridization outperforms the combinations of MA with other local search algorithms. Abraham et al. [26] proposed the variable neighborhood particle swarm optimization algorithm. They empirically showed the performance of the proposed algorithm and its feasibility and effectiveness for scheduling work flow applications. Lusa and Potts [27] proposed the VNS algorithm for the constrained task allocation problem and compared the performance of the proposed algorithm with the other local search procedures. Moghaddam et al. [28] presented a hybrid GA and VNS to reduce the overall cost of task executions in grid environment.

Few works have considered the optimization of makespan and flowtime objectives for the scheduling problem [25,29]. Jacob et al. [30] studied the optimization of four objectives, namely makespan, resource utilization, time and cost of application for solving the HCSP. Xu et al. [31] experimented the Chemical Reaction Optimization (CRO) algorithm based grid job scheduling problem by considering makespan, flowtime and tardiness of the solution.

The VNS algorithm has received relatively little attention in solving the grid job scheduling problem. From the literature, it is known that VNS has been used in hybridization with other algorithms for such problems. To our knowledge, there are no other antecedents on applying explicit VNS to solve the heterogeneous computing scheduling problem tackled in this work, so the approach presented here is a novel approach in

this line of research. The performance of VNS algorithm depends on the performance of its neighborhood. Working in that line, the performance of VNS algorithm had been enriched by framing new problem specific neighborhoods, in order to solve the large scale heterogeneous computing scheduling problem instances. The multiobjective version of the scheduling problem studied in this work considers the optimization of an aggregation function that sums the makespan and flowtime of the solutions.

3. The Grid scheduling process and components

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities [1]. A Grid Scheduler (GS) receives applications from grid users, selects feasible nodes for these applications according to the acquired information from the Grid Information Service (GIS) module, and finally generates application-to-node mappings, based on certain objective functions and predicted node performance. Scheduling algorithms are used in the GS for mapping tasks to resources in order to simultaneously optimize both systems-related (e.g. makespan) and user-related objectives (e.g. flowtime).

Fig. 1 depicts a model of grid scheduling system. Grid scheduler is referred as Meta scheduler in the literature [32,33] and which is not an indispensible component in the Grid infrastructure.

The role of the Grid information service is to provide information about the status of available nodes to Grid schedulers. GIS is responsible for collecting and predicting the node state information, such as CPU capacities, memory size, network bandwidth, software availabilities and load of a site in a particular period. GIS can answer queries for node information or push information to subscribers.

Besides raw node information from GIS, application properties such as approximate instruction quantity, memory and storage requirements, subtask dependency in a job and communication volumes and performance of a node for different application species are also necessary for making a feasible schedule. Application profiling (AP) is used to extract properties of applications, while analogical benchmarking (AB) provides a measure of how well a node can perform a given type of job [34,35]. Cost estimation module computes the cost of candidate schedules. On the basis of knowledge from AP, AB and cost estimation module, the scheduler chooses those that can optimize the objective functions.

The Launching and Monitoring (LM) module is known as the ''binder'' which implements a finally-determined schedule by submitting applications to selected nodes, staging input data and executables if necessary, and monitoring the execution of the applications [36].

A Local Resource Manager (LRM) is mainly responsible for two jobs: local scheduling inside a node domain, where not only jobs from exterior Grid users, but also jobs from the domain's local users are executed, and reporting node information to GIS. For clarity, some key terminologies [37] are defined as follows.

- **Grid node**

A grid node is an autonomous entity composed of one or multiple nodes. The computational capacity of the node depends on its number of CPUs, amount of memory, basic storage space and other specifications.

- **Jobs and operations**

A job is considered as a single set of multiple atomic operations/tasks. Each operation will be typically allocated to execute on one single node without pre-emption. It has input and output data and processing requirements in order to complete its task.

- **Task scheduling**

A task scheduling is the mapping of tasks to a selected group of nodes which may be distributed in multiple administrative domains.

This work deals with the static scheduling problem, in which all tasks can be independently performed. All the information about tasks and resources is gathered by the Grid scheduler before computing the schedule, and the task to resource assignment is not allowed to change during the execution. Static scheduler acts as the basic building block to develop a powerful dynamic scheduler, able to solve more complex scheduling problems. The concept of static scheduling frequently appears in many scientific research problems, especially in Single-Program Multiple-Data applications used for multimedia processing, scientific computing, data mining, parallel domain decomposition of numerical models for physical phenomena, etc. The independent tasks model also arises when different users submit their tasks to execute in volunteer-based and grid computing services and in parameter sweep applications, which are structured as a set of multiple experiments, each one executed with a different set of parameter values [21].

4. Scheduling problem formulation

The problem is formulated based on the ''Expected time to compute'' (ETC) model [16]. In a particular time interval, n independent jobs $J_1, J_2, J_3, \ldots, J_n$ (expressed in millions of instructions) are submitted to Meta scheduler for scheduling, and at the same time, GIS locates m (usually $n \gg m$) grid nodes $G_1, G_2, G_3, \ldots, G_m$, donating nodes. The processing power of a grid node is measured in terms of ''millions of instructions per second''. To address the problem, the following assumptions are considered [31].

1. Any job J_i has to be processed in one of the grid nodes G_j until completion.
2. Jobs come in batch mode.
3. A node cannot remain idle when jobs have been assigned to it.
4. A job can only be executed on one grid node in each interval.

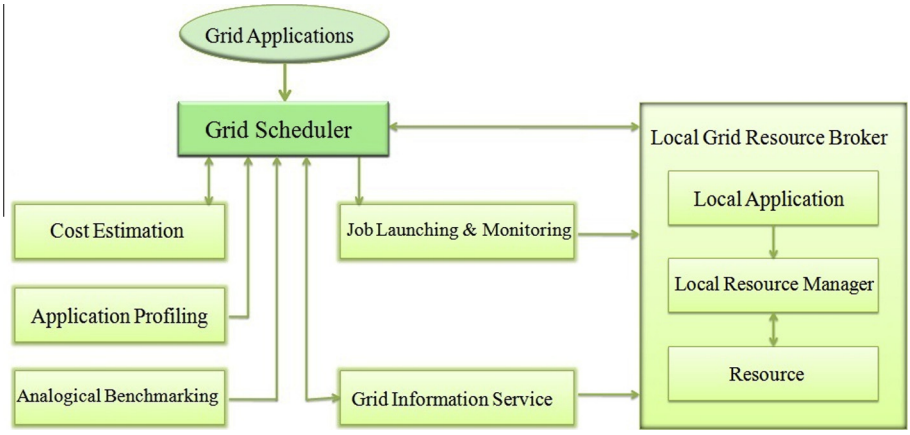

Figure 1 A logical Grid scheduling architecture.

5. When a node processes its tasks, there is no priority distinctions between the tasks assigned in the previous intervals and those assigned in the current interval.

Based on the specifications of the nodes and tasks, Meta scheduler computes $n \times m$ matrix ETC ($ETC: J \times G \rightarrow R^{+}$) where entity ETC_{ij} represents the expected time for node j to process job i. R^{+} denotes that the entry ETC_{ij} is the positive real number. The multi-objective scheduling problem can be formulated by defining the following notations and variables.

i index of tasks, $i = 1, 2, \ldots, n$,
j index of nodes, $j = 1, 2, \ldots, m$,
n number of tasks,
m number of heterogeneous nodes,
x_i variable representing the node to execute the task i,
$x_i^{(U)}$ maximum allowed value of x_i,
$x_i^{(L)}$ minimum allowed value of x_i,
ETC_{ij} expected time for node j to process task i,
C_j completion time of node j.

The job scheduling problem is considered as a bi-objective optimization problem, in which both makespan and flowtime are simultaneously minimized, which is possible since both parameters are measured in the same unit (time units).

$$fitness = a \times Makespan + (1 - a) \times \left(\frac{Flowtime}{m}\right) \tag{1}$$

As flowtime has higher order of magnitude over makespan, it is normalized by m. Actually, in this method the multi-objective task scheduling problem is converted to a single objective scheduling problem using the linear combination of both objectives. The objective function can be expressed as follows:

$$\text{Minimize fitness, } f(x) = a * max\left\{\sum_{[i/x_i=j]} ETC_{ij}\right\} + \frac{(1-a)}{m} * \left(\sum_{j=1}^{m}\left(\sum C_j\right)\right) \tag{2}$$

$$\text{s.t.} \quad x = \{x_1, x_2, \ldots, x_n\},\ \forall x_i \in [1, m],\ \forall i \in [1, n],\ \forall j \in [1, m] \tag{3}$$

$$a = 0.75 \tag{4}$$

$$ETC_{ij} > 0, \quad i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, m \tag{5}$$

$$x_i^{(U)} = m, \quad i = 1, 2, \ldots, n \tag{6}$$

$$x_i^{(L)} = 1, \quad i = 1, 2, \ldots, n \tag{7}$$

$$x_i^{(U)} \geqslant x_i \geqslant x_i^{(L)}, \quad i = 1, 2, \ldots, n \tag{8}$$

$$C_j = \sum_{[i/x_i=j]} ETC_{ij}, \quad [i/x_i = j]$$

represents the tasks assigned to node j (9)

In this model, the objective function (2) minimizes both makespan and flowtime. Constraint (3) defines the weighing factor [25]. Constraint (4) denotes a vector composed of n objective function parameters. Constraint (5) ensures that all entries of $n \times m$ ETC matrix are positive. Constraints (6) and (7) define the upper and lower boundary constraints of the objective function parameters respectively. Constraint (8) defines the upper and lower boundary constraints of the variable x_i. Constraint (9) calculates the completion time of node j, which is defined as the time required for node j to complete all its assigned tasks.

5. Implementation of MVNS algorithm for scheduling jobs on computational grid

The following subsections deal with the representation of solution, generation of initial solution, explanation of neighborhood structures, and the proposed grid job scheduling algorithm.

5.1. Solution representation

The solution is represented as an array of length equal to the number of jobs. The value corresponding to each position i

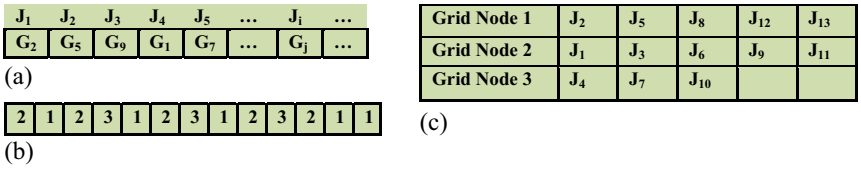

Figure 2 (a) Solution representation, (b) solution for the problem of 13 jobs and 3 Grid nodes, (c) mapping of jobs with Grid nodes for the solution given in (b).

in the array represents the node to which job i was allocated. The representation of the solution for the problem of scheduling 13 jobs to 3 Grid nodes is illustrated in Fig. 2. The first element of the array denotes the first job (J_1) in a batch which is allocated to the Grid node 2; the second element of the array denotes the second job (J_2) which is assigned to the Grid node 1, and so on.

5.2. Initial solution generation

The random initial solution is considered. Let x be the solution composed of n parameters, which is specified as $x = \{x_1, x_2, \ldots, x_n\}$. The parameters are subject to lower and upper boundary constraints (Eqs. 3, 6, 7 and 8).

5.3. Neighborhood structures

The neighborhood structure defines the type of modifications a current solution can undergo and thus, different neighborhoods offer different ways to explore the solution space. In other words, definition of the proper neighborhood structures leads to better exploration and exploitation of the solution space. Two attributes of the solutions are considered to define six neighborhood structures so that a larger part of the solution space can be searched and the chance of finding good solutions will be enhanced. The attributes that can be altered from one solution to another are "Random assignment of grid nodes to jobs", and "Workload of grid nodes". The defined neighborhood structures and corresponding moves associated with them are explained in detail below.

5.3.1. SwapMove

This neighborhood structure provides a set of neighbors for current solution x, based on exchanging the nodes assigned for the randomly selected three jobs.

5.3.2. Makespan-InsertionMove

This neighborhood assigns the *Light* node to the randomly selected job in the job list of *Heavy* node. *Light* and *Heavy* nodes are the nodes with minimum and maximum local makespan respectively, where the local makespan of individual node gives the completion time of its latest job. Maximum local makespan is the makespan of the solution.

5.3.3. InsertionMove

Neighbors generated using this neighborhood structure can be constructed using the assignment of random node G_1 in G to the random job J_1 in J.

5.3.4. Weightedmakespan-InsertionMove

Based on this neighborhood structure, solutions are generated by assigning the random node Lr to the random job J_1 selected from the job list of the random node Hr. Lr and Hr are the nodes having local makespan value less than or equal to 0.25 and greater than or equal to 0.75 of the makespan of current solution respectively.

5.3.5. BestInsertionMove

This neighborhood maps the longest job J_1 in the job list of *Heavy* to the node having minimum execution time for J_1.

To illustrate, a small scale job scheduling problem involving 3 nodes and 13 jobs is considered. The node speeds are 4, 3, 2 cycles/second, and the job lengths of 13 jobs are 6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, 60 cycles, respectively. Consider the initial solution with fitness 100.055, which is represented in Fig. 3a. The SwapMove operator swaps the nodes assigned for the selected three jobs J_9, J_2, and J_4 (already mapped with G_3, G_2, and G_1 respectively) and changes the fitness of the solution as 92.33 (Fig. 3b). Then the job J_1 assigned for G_3 (*Heavy*-with localmakespan 105) is mapped with the node G_2 (*Light*-with localmakespan 28), according to the Makespan-InsertionMove neighborhood. Thus the fitness of the current solution becomes 90, which is illustrated in Fig. 3c. Then InsertionMove neighborhood selects the node G_2 and maps with the Job J_{11} (already mapped with G_1). This mapping changes the localmakespan of G_1 and G_2 (18 and 46 respectively), and also the fitness of current solution as 90.16 (Fig. 3d). According to the Weightedmakespan-InsertionMove, the job J_{13} from the joblist of G_3 (considered as Hr) is assigned to the node G_1 (considered as Lr). This neighborhood minimizes the fitness of current solution as 66.58 (Fig. 3e). Then the BestInsertionMove neighborhood selects the longest job J_{12} from G_3 (considered as *Heavy*) and assigns with G_1 (High speed node of J_{12}) (Fig. 3f). Hence the final solution has the fitness 46, which is the optimal result for the example problem.

5.4. Proposed MVNS grid job scheduling algorithm

VNS is a metaheuristic which systematically exploits the idea of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. The term VNS is referred to all local search based approaches that are centered on the principle of systematically exploring more than one type of neighborhood structures during the search. VNS iterates over more than one neighborhood structures until some stopping criterion is met. The basic scheme of the VNS was proposed by Mladenović and Hansen [38]. Its advanced principles for solving combinatorial optimization problems and applications were further introduced in [39–41] and recently in [42].

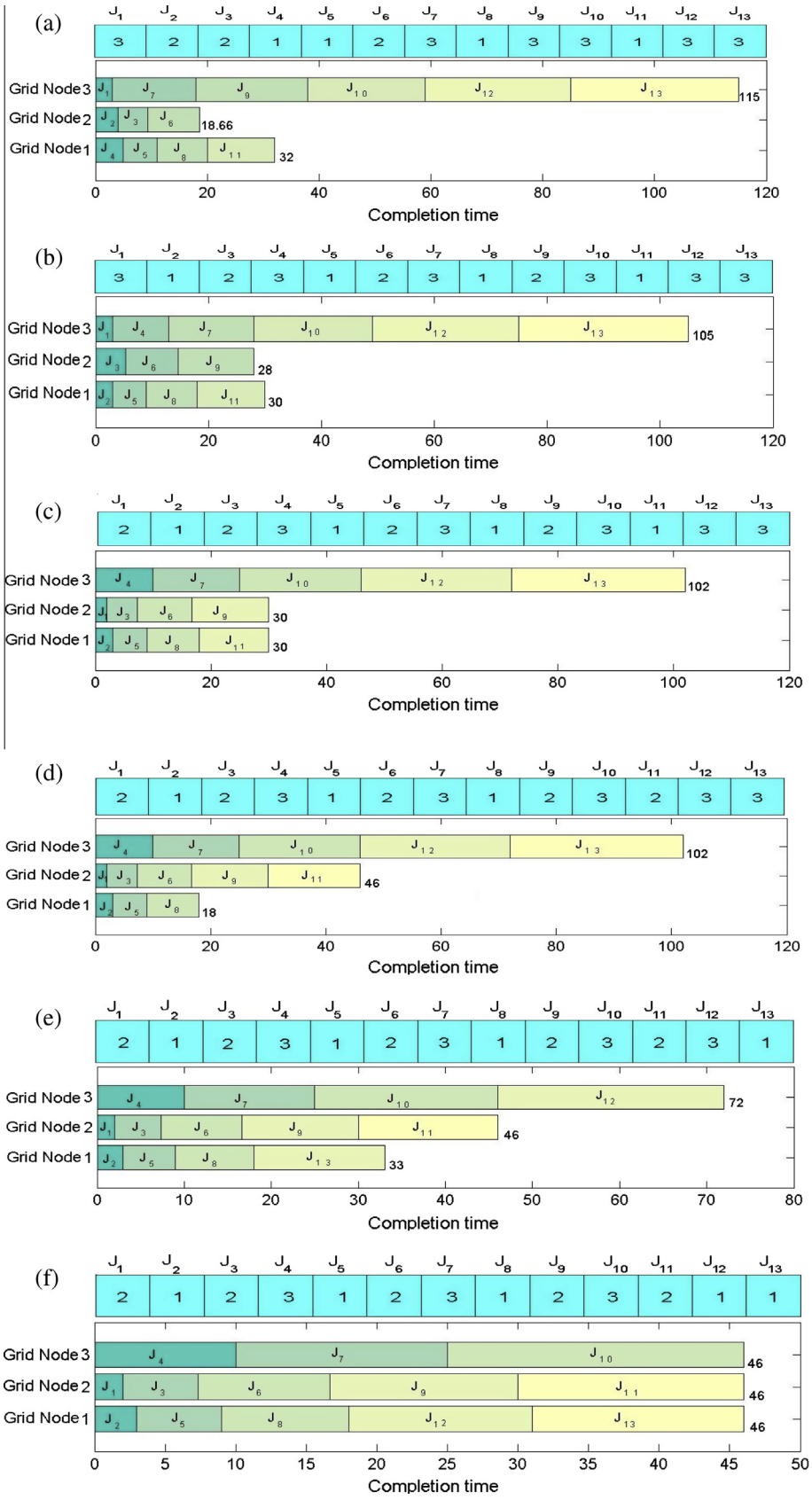

Figure 3 Explanation of different neighborhood structures (a) initial solution, (b) SwapMove, (c) Makespan-InsertionMove, (d) InsertionMove, (e) Weightedmakespan-InsertionMove, and (f) BestInsertionMove.

VNS uses a finite set of pre-selected neighborhood structures denoted as $N_k(k = 1, \ldots, k_{max})$. $N_k(x)$ denotes the set of solutions in the kth neighborhood of solution x. VNS employs a local search to obtain a solution $x \epsilon X$, called as a local minimum, such that there exists no solution $x' \epsilon N_k(x) \subseteq X$ with $f(x') < f(x)$. The local search can be performed in different ways. The generic way consists of choosing an initial solution x, finding a direction of descent from x within a neighborhood $N(x)$, and moving to the minimum of $f(x)$ within $N(x)$ in the same direction. If there is no direction of descent, the heuristic stops; otherwise, it is iterated. After the local search, a change in the neighborhood structure is performed. Function NeighborhoodChange compares the value $f(x')$ of a new solution x' with the value $f(x)$ of the incumbent solution x obtained in the neighborhood k. If an improvement is obtained, k is returned to its initial value and the incumbent solution is updated with the new one. Otherwise, the next neighborhood is considered.

The proposed MVNS grid job scheduling algorithm is summarized in Algorithm 1. VNS uses two parameters: t_{max}, which is the maximum CPU time allowed as the stopping condition, and k_{max}, which is the number of neighborhood structures used. Step 4 of Algorithm 1, which is called shaking, randomly chooses a solution x' from the k^{th} neighborhood of the incumbent solution x. After improving this solution via the PALSheuristic local search (Algorithm 3), a neighborhood change is employed. The fitness of the solution is evaluated based on the procedure described in the Algorithm 2.

Algorithm 1. MVNS grid job scheduling algorithm

Input: x, k_{max}, t_{max}, $ETC[\,][\,]$, $PALS\_maxiter$
Output: x
1 ***repeat***
2 $k \leftarrow 1$
3 ***repeat***
4 $x' \leftarrow shake(x, k)$
5 $x'' \leftarrow PALSHeuristic(x', ETC[\,][\,], PALS\_maxiter)$ /* Local search */
6 $Neighborhoodchange(x, x'', k)$
7 ***until*** $k = k_{max}$
8 $t \leftarrow Cputime()$
9 ***until*** $t > t_{max}$

Algorithm 2. $f(x, ETC[\,][\,])$ /*fitness evaluation */

Input: $x, ETC[\,][\,]$
Output: $localmakespan[\,], fitnes$
1. $[n, m] \leftarrow size(ETC[\,][\,])$
2. $localmakespan[\,] \leftarrow 0$; $flowtime \leftarrow 0$
3. ***for*** $i = 1$ to n ***do***
4. $localmakespan(x[i]) \leftarrow localmakespan(x[i]) + ETC(i, x[i])$
5. ***endfor***
6. $Makespan \leftarrow maximum(localmakespan[\,])$
7. ***for*** $j = 1$ to m ***do***
8. $flowtime \leftarrow flowtime + localmakespan[j]$
9. ***endfor***
10. $fitnes \leftarrow 0.75 * Makespan + 0.25 * (flowtime/m)$

Algorithm 3. Problem Aware Local Search Heuristic

Input: x, $ETC[\,][\,]$, $PALS\_maxiter$
Output: x
1 ***for*** $i = 1$ to $PALS\_maxiter$ ***do***
2 $[localmakespan[\,], fitnes] \leftarrow f(x, ETC[\,][\,])$ / * Algorithm 2 */
3 $Best \leftarrow \infty$
4 $JJ[\,] \leftarrow$ Job list of $Heavy$
5 Select a random node G_1, where $G_1 \neq Heavy$
6 $JJJ[\,] \leftarrow$ Job list of G_1
7 $ln \leftarrow length(JJ[\,])$
8 $lnr \leftarrow length(JJJ[\,])$
9 $startheavy \leftarrow randi(1, ln - 1)$
10 $endheavy \leftarrow randi(startheavy, ln)$
11 $startres \leftarrow randi(1, lnr - 1)$
12 $endres \leftarrow randi(startres, lnr)$
13 ***for*** $i = startheavy$ to $endheavy$ **do**
14 ***for*** $j = startres$ to $endres$
15 $x'' \leftarrow$ Swap the resources assigned for $JJ[i]$ and $JJJ[j]$
16 $[localmakespan[\,], temp] \leftarrow f(x, ETC[\,][\,])$
17 **if** $(temp < Best)$
18 $x' \leftarrow x''$
19 $Best \leftarrow temp$
20 ***endif***
21 ***endfor***
22 ***endfor***
23 ***if*** $(fitnes > Best)$ ***then***
24 $fitnes \leftarrow Best$
25 $x \leftarrow x'$
26 ***break***
27 ***endif***
28 ***endfor***

5.4.1. Problem Aware Local Search (PALS)

Basic concept of this local search has been used in the literature for the DNA fragment assembly problem [43], and the heterogeneous computing scheduling problem [44]. Working on a given schedule x, this algorithm selects a node *Heavy* to perform the search. The outer cycle iterates on '*it*' number of jobs (where $it = endheavy - startheavy + 1$) of the node *Heavy*, while the inner cycle iterates on '*jt*' number of jobs (where $jt = endres - startres + 1$) of the randomly selected node G_1, other than *Heavy*. For each pair (i,j), the double cycle calculates the makespan variation when swapping the nodes assigned for $JJ[i]$ and $JJJ[j]$, where JJ and JJJ denote the job list of the nodes *Heavy* and G_1 respectively. This neighborhood stores the best improvement on the makespan value for the whole schedule found in the evaluation process of $it \times jt$. At the end of the double cycle, the best move found so far is applied. In this algorithm, *startheavy* and *endheavy*, *startres* and *endres* are assigned with random values based on the length of array JJ and JJJ respectively (Refer lines 4 and 6 of Algorithm 3). The randomness introduced in the parameters *endheavy* and *endres* makes this local search to differ from the concept existing in the literature.

After the extensive experimentation, the combination of SwapMove, Makespan-InsertionMove and BestInsertionMove was selected for the proposed MVNS. The details of the neighborhood structures are given in Algorithms 4, 5, and 6. The

parameters *PALS_maxiter* and k_{max} are set to 5 and 3 respectively.

Algorithm 4. SwapMove

Input: x
Output: x'
1. Choose three random jobs J_1, J_2, and J_3 in J
2. $x' \leftarrow$ Swap the resources assigned for J_1, J_2, and J_3 of x

Algorithm 5. Makespan-InsertionMove

Input: x, $ETC[\,][\,]$
Output: x'
1. Evaluate the fitness of x
2. Select two nodes *Light* and *Heavy* from G, where *Light* and *Heavy* are the nodes with minimum and maximum localmakespan respectively
3. Select a random job J_1 from the job list assigned for *Heavy*
4. $x' \leftarrow$ Assign *Light* to J_1

Algorithm 6. BestInsertionMove

Input: x, $ETC[\,][\,]$
Output: x'
1. Evaluate the fitness of x
2. Select a longest job from the job list assigned for *Heavy*
3. Select a node G_1 in G, where G_1 has minimum execution time for J_1
4. $x' \leftarrow$ Assign G_1 to J_1

6. Computational experiments

When facing the heterogeneous computing scheduling problem, researchers have often used the test instances proposed by Braun et al. [16], following the *ETC* performance estimation model by Ali et al. [45]. *ETC* takes into account three key properties: machine heterogeneity, task heterogeneity, and consistency. Machine heterogeneity evaluates the variation of execution times for a given task across the heterogeneous computing nodes, while task heterogeneity represents the variation of the tasks execution times for a given machine. Regarding the consistency property, in a consistent scenario, whenever a given node G_j executes any task J_i faster than other machine G_k, and then node G_j executes all tasks faster than machine G_k. In an inconsistent scenario, a given machine G_j may be faster than machine G_k when executing some tasks and slower for others. Finally, a semiconsistent scenario models those inconsistent systems that include a consistent subsystem.

Nesmachnow et al. [21] proposed a test suite of several large dimension heterogeneous computing scheduling problem instances, in order to model large heterogeneous computing clusters and medium sized grid infrastructures. All test instances of each dimension are composed of m grid nodes and n jobs, which is referred as the configuration $m \times n$. Each dimension has 24 test instances regarding all the heterogeneity and consistency combinations, twelve of them considering the parameterization values from Ali et al. [45], and twelve using the values from Braun et al. [16]. The instances are named as *M.u_x_yyzz*, where the first letter (*M*) describes the heterogeneity model (A for Ali, and B for Braun), *u* means uniform distribution (in the *ETC* matrix generation), *x* is the type of consistency (*c* – consistent, *i* – inconsistent and *s* means semi-consistent), and *yy* and *zz* indicate the job and machine heterogeneity (hi – high, and lo – low).

This paper considers the test instances proposed by Nesmachnow et al., with dimension 1024×32, 2048×64, and 4096×128 [21]. The grid job scheduling algorithm was developed using MATLAB R2010a and run on an Intel(R) Core(TM) i5 2.67 GHz CPU with 4 GB RAM. As the problem size increases, the evaluation of the fitness function consumes larger computing time than the application of neighborhood operators. The maximum running time of the algorithm is not set to uniform value for all configurations. The stopping condition t_{max} is set to 150, 300, and 700 s for 1024×32, 2048×64, and 4096×128 dimension problems respectively.

6.1. Results and discussion

This section discusses the experimental results of applying the MVNS algorithm to solve the grid job scheduling problem. The MVNS results are compared with the deterministic heuristic Min–Min algorithm, Simulated Annealing algorithm and GRASP algorithm. For SA, Initial temperature, temperature reduction factor and reannealing interval are set to 50, 0.95 and 10 respectively. GRASP was experimented with PALS heuristic (Algorithm 3) in the local search phase, in which *PALS_maxiter* and threshold parameter are set to 50 and 0.2 respectively.

Each experiment (for each algorithm) was repeated 50 times with different random seeds. The fitness values of the best solutions throughout the optimization run were recorded. In the computation experiments, 72 test instances were solved with SA, GRASP, Min–Min algorithm and MVNS algorithm. The experimental results displayed in bold fonts indicate that the corresponding solution is the best solution obtained out of all algorithms considered for comparison along with MVNS algorithm. The overall best result produced by the MVNS algorithm compared with all algorithms is represented in bold and italic.

The improvement of an algorithm over another is computed using Eq. (10).

$$\textit{Improvement}\ (\%) = \frac{\delta_1 - \delta_2}{\delta_2} \times 100\% \tag{10}$$

where δ_1 and δ_2 are the fitness values of two different algorithms.

6.1.1. Solution quality

Tables 1–3 show the MVNS result for 1024×32, 2048×64, and 4096×128 configurations. The best, average, and standard deviation on the fitness results achieved during the experimentation of Min–Min, SA, GRASP and MVNS algorithm are reported in Tables 1–3. From Tables 1–3, it is observed that MVNS produces a good quality schedule for all the test

Table 1 Fitness results for the test instances of 1024×32 configuration.

Instance	Min–Min	S A			GRASP			MVNS			Impr. over (%)		
		Best	Average	σ (%)	Best	Average	σ (%)	Best	Average	σ (%)	Min–Min	SA	GRASP
A.u_c_hihi	32303475.0	49691000.0	52535501.9	0.23	59296276.0	60246004.0	0.17	22371956.0	22816668.0	0.12	30.74	54.98	62.27
A.u_c_hilo	3202078.0	5024400.0	5256028.4	0.18	5804483.0	5868504.5	0.12	2239553.3	2329903.8	0.11	30.06	55.43	61.42
A.u_c_lohi	2999.5	5744.9	5801.1	0.25	5707.0	5761.8	0.18	2094.2104	2138.5461	0.14	30.18	63.54	63.30
A.u_c_lolo	322.2	1023.0	1055.4	0.21	587.5	594.7	0.21	225.4	233.2	0.15	30.04	77.97	61.63
A.u_i_hihi	7401064.0	40800000.0	45610922.1	0.13	60450692.0	61488296.0	0.19	5377229.5	5448493.5	0.17	27.35	86.82	91.10
A.u_i_hilo	697545.9	3781400.0	4390844.7	0.15	5738513.5	5813609.0	0.18	508015.1	513903.4	0.13	27.17	86.57	91.15
A.u_i_lohi	738.4	4291.0	4423.9	0.11	5759.698.5	5824.558.5	0.20	506.9	507.4	0.14	31.34	88.18	91.19
A.u_i_lolo	70.9	870.3	876.4	0.17	600.9	609.7	0.16	51.8	52.4	0.20	26.94	94.05	91.37
A.u_s_hihi	18596806.0	51040000.0	51572046.8	0.16	60811624.0	61591852.0	0.16	13564793.0	13928393.0	0.10	27.06	73.42	77.69
A.u_s_hilo	1794747.0	4817900.0	5140694.7	0.20	5992918.0	6092478.0	0.15	1314495.6	1320775.0	0.17	26.76	72.72	78.07
A.u_s_lohi	1775.5	5843.8	5900.2	0.17	5893.8	5938.9	0.19	1344.5	1379.6	0.09	24.27	76.99	77.19
A.u_s_lolo	189.6	931.9	942.4	0.19	610.1	619.4	0.22	136.3	137.6	0.11	28.12	85.38	77.66
B.u_c_hihi	9337642.0	15579000.0	15792000.0	0.14	17502926.0	17579946.0	0.10	6782008.0	6801153.0	0.10	27.37	56.47	61.25
B.u_c_hilo	96095.4	148340.0	160230.0	0.10	177707.3	180220.3	0.14	67642.0	69478.0	0.12	29.61	54.40	61.94
B.u_c_lohi	328379.2	516070.0	531230.0	0.19	601621.1	612066.1	0.13	237388.9	239105.3	0.11	27.71	54.00	60.54
B.u_c_lolo	3356.1	5945.7	6076.2	0.16	5922.2	6086.5	0.19	2368.3	2410.0	0.09	29.43	60.17	60.01
B.u_i_hihi	2430464.0	12067000.0	12998000.0	0.11	17947406.0	18234360.0	0.11	1696470.3	1734982.0	0.16	30.19	85.94	90.55
B.u_i_hilo	21965.5	123700.0	135330.0	0.17	177127.2	178480.8	0.13	15521.0	15626.0	0.15	29.34	87.45	91.23
B.u_i_lohi	71860.4	448320.0	463540.0	0.12	576534.1	589052.8	0.12	50793	50846.0	0.13	29.32	88.67	91.19
B.u_i_lolo	727.9	5650.1	5728.5	0.19	5692.9	5787.9	0.14	509.0	512.6	0.08	30.07	90.99	91.06
B.u_s_hihi	5288315.0	14570000.0	14804000.0	0.20	17642298.0	17837298.0	0.17	3780893.5	3806248.3	0.12	28.50	74.05	78.57
B.u_s_hilo	54543.1	151830.0	157430.0	0.13	179943.6	182090.1	0.16	40931.0	40994.0	0.13	24.96	73.04	77.25
B.u_s_lohi	173757.7	493960.0	511880.0	0.15	579778.4	585129.1	0.22	128936.9	129450.7	0.10	25.79	73.89	77.76
B.u_s_lolo	1841.8	5706.3	5824.9	0.18	5812.3	6014.0	0.17	1324.8	1395.2	0.18	28.07	76.78	77.21

Table 2 Fitness results for the test instances of 2048×64 configuration.

Instance	Min–Min	S A			GRASP			MVNS			Impr. over (%)		
		Best	Average	σ (%)	Best	Average	σ (%)	Best	Average	σ (%)	Min–Min	SA	GRASP
A.u_c_hihi	28033987.0	63057210.0	65655774.9	0.20	75062880.0	75076790.0	0.14	21029586.0	21262348.0	0.12	24.99	66.65	71.98
A.u_c_hilo	2705653.0	6211341.2	6454726.9	0.14	7266130.5	7292718.5	0.17	2022377.5	2040942.9	0.13	25.25	67.44	72.17
A.u_c_lohi	2814.2	8309.7	8439.8	0.12	7403.7	7555.1	0.16	2063.1	2108.7	0.11	26.69	75.17	72.13
A.u_c_lolo	274.4	1148.9	1170.1	0.17	726.2	797.3	0.12	202.6	216.7	0.18	26.17	82.37	72.10
A.u_i_hihi	3743631.0	62529503.6	63917261.3	0.16	70379080.0	70751824.0	0.17	2759628.3	2769008.0	0.14	26.28	95.59	96.08
A.u_i_hilo	398833.4	6286869.7	6364513.8	0.18	7078694.0	7110455.5	0.14	296976.4	300144.7	0.18	25.54	95.28	95.80
A.u_i_lohi	376.4	7372.9	7451.7	0.19	7077.3	7134.3	0.11	288.9	299.4	0.16	23.24	96.08	95.92
A.u_i_lolo	39.4	915.8	947.3	0.13	715.4	798.0	0.10	29.2	32.4	0.14	25.94	96.81	95.92
A.u_s_hihi	15976701.0	63710491.2	64741290.5	0.18	72879208.0	73588240.0	0.15	11702970.0	12037198.0	0.19	26.75	81.63	83.94
A.u_s_hilo	1401827.0	6397732.1	6479820.2	0.14	7048720.5	7144016.5	0.17	1092334.1	1117282.5	0.12	22.07	82.92	84.50
A.u_s_lohi	1470.237.0	7443.3	7665.6	0.15	7125.5	7246.6	0.20	1142.9	1216.9	0.11	22.26	84.65	83.96
A.u_s_lolo	156.4123.0	1011.1	1092.6	0.19	725.6	793.9	0.21	120.4	183.8	0.17	23.02	88.09	83.41
B.u_c_hihi	8121108.0	18158549.2	18376320.1	0.21	21944496.0	22045496.0	0.19	5989006.0	6091842.5	0.13	26.25	67.02	72.71
B.u_c_hilo	86169.9	186526.4	193921.4	0.20	223248.7	234241.1	0.13	64344.0	65238.0	0.14	25.33	65.50	71.18
B.u_c_lohi	275896.6	610010.6	620614.3	0.15	722534.5	728163.1	0.17	204034.09	204913.9	0.19	26.05	66.55	71.73
B.u_c_lolo	2841.1	7780.9	8175.8	0.13	7419.4	7512.7	0.15	2097.5	2154.9	0.17	26.17	73.04	71.73
B.u_i_hihi	1165077.0	18452110.1	19626383.8	0.10	21808722.0	21901420.0	0.18	835526.0	849462.0	0.11	28.29	95.47	96.17
B.u_i_hilo	11379.9	179072.2	191441.1	0.16	219308.0	224996.6	0.13	8610.1	8737.5	0.15	24.34	95.19	96.07
B.u_i_lohi	39146.1	619006.4	651141.9	0.19	729385.9	743063.9	0.11	29719.0	30775.0	0.13	24.08	95.20	95.93
B.u_i_lolo	402.7	7226.8	7342.4	0.14	7060.0	7129.1	0.15	287.0	318.4	0.12	28.73	96.03	95.93
B.u_s_hihi	4585171.0	17836001.2	19109691.2	0.15	21312070.0	21347198.0	0.09	3512196.5	3539540.8	0.18	23.40	80.31	83.52
B.u_s_hilo	46261.4	185801.3	195186.9	0.17	218541.4	219074.1	0.17	36135.0	37281.0	0.11	21.89	80.55	83.47
B.u_s_lohi	155270.5	588907.6	610393.4	0.12	715308.9	721908.3	0.13	117174.3	127560.3	0.16	24.54	80.10	83.62
B.u_s_lolo	1575.4	7552.6	7617.3	0.11	7238.8	7373.8	0.15	1225.0	1340.9	0.15	22.24	83.78	83.08

Table 3 Fitness results for the test instances of 4096×128 configuration.

Instance	Min–Min	S A			GRASP			MVNS			Impr. over (%)		
		Best	Average	σ (%)	Best	Average	σ (%)	Best	Average	σ (%)	Min–Min	SA	GRASP
A.u_c_hihi	24593149.0	79148329.6	80748803.4	0.14	89394336.0	89424336.0	0.13	19487692.0	19663084.0	0.11	20.75	75.37	78.20
A.u_c_hilo	2424948.0	8033663.2	8128561.1	0.16	8823599.0	8862629.0	0.12	1922381.6	1931903.0	0.15	20.72	76.07	78.21
A.u_c_lohi	2470.0	9874.8	9997.8	0.17	8629.3	8765.4	0.11	1920.4	2000.6	0.14	22.25	80.55	77.75
A.u_c_lolo	244.3	1266.2	1370.0	0.11	885.9	900.7	0.09	192.4	234.0	0.11	21.24	84.80	78.28
A.u_i_hihi	1839977.0	72457718.4	73869752.8	0.19	80649928.0	81731023.1	0.13	1511832.1	1543606.4	0.16	17.83	97.91	98.12
A.u_i_hilo	201020.1	7375050.2	7475654.4	0.13	8112595.0	8147229.5	0.16	159347.4	165785.2	0.12	20.73	97.83	98.03
A.u_i_lohi	195.2	8695.4	8797.1	0.20	7895.4	8052.7	0.17	157.4	181.5	0.13	19.37	98.18	98.00
A.u_i_lolo	20.2	998.0	1106.1	0.17	816.1	898.0	0.19	16.6	20.7	0.09	17.69	98.33	97.96
A.u_s_hihi	12527613.0	74430768.8	75017361.9	0.16	82881328.0	83730163.0	0.16	11131578.0	11284319.0	0.11	11.14	85.04	86.57
A.u_s_hilo	1286190.0	7494309.5	7514031.1	0.12	8291316.0	8319271.4	0.13	1148052.3	1169810.3	0.14	10.74	84.68	86.15
A.u_s_lohi	1310.1	8947.3	9329.4	0.11	8273.7	8376.6	0.11	1147.7	1227.1	0.11	12.39	87.17	86.12
A.u_s_lolo	131.2	1066.6	1113.7	0.12	835.2	899.4	0.17	118.9	140.8	0.13	9.34	88.85	85.76
B.u_c_hihi	7590400.0	24228415.1	24616949.9	0.13	27585398.0	27991308.1	0.12	5918685.5	5985981.0	0.10	22.02	75.57	78.54
B.u_c_hilo	72638.3	245304.8	253107.4	0.18	262315.0	265552.4	0.14	57197.6	58042.1	0.11	21.25	76.68	78.19
B.u_c_lohi	249037.0	778626.7	807824.1	0.12	875624.0	889914.3	0.19	195185.1	197643.5	0.13	21.62	74.93	77.71
B.u_c_lolo	2441.0	9566.3	9725.4	0.16	8853.5	8900.1	0.15	1950.6	1201.9	0.11	20.09	79.60	77.96
B.u_i_hihi	584618.6	21902051.1	22263576.9	0.14	24223036.0	24295410.3	0.16	471521.6	477823.7	0.13	19.35	97.85	98.05
B.u_i_hilo	6321.1	222209.2	225070.2	0.15	242991.9	250100.1	0.12	4945.9	5034.2	0.12	21.76	97.77	97.96
B.u_i_lohi	18749.9	743870.4	747313.8	0.12	803492.7	809431.1	0.17	15613.9	16148.3	0.14	16.72	97.90	98.05
B.u_i_lolo	203.9	8526.3	8921.9	0.19	8048.1	8200.1	0.13	162.5	203.4	0.12	20.30	98.09	97.98
B.u_s_hihi	3797274.0	22403862.4	23761526.7	0.11	25001098.0	25151943.2	0.12	3463622.5	3478560.3	0.11	8.78	84.54	86.14
B.u_s_hilo	39325.8	218693.9	229336.4	0.09	250195.5	258700.1	0.11	35569.2	36890.9	0.13	9.55	83.74	85.79
B.u_s_lohi	130279.3	758302.0	769156.7	0.11	850445.5	859920.1	0.17	117944.9	119264.3	0.11	9.46	84.44	86.13
B.u_s_lolo	1332.2	8610.2	8766.8	0.16	8244.0	8903.1	0.16	1180.9	1227.4	0.15	11.35	86.28	85.67

instances. Min–Min algorithm gave the second best mapping by yielding better solutions for all the test instances. As the numbers of jobs and resources increase, the performance of Min–Min algorithm improves significantly. SA and GRASP yielded the better mapping for the consistent test cases. The percentage improvement of MVNS over Min–Min, SA and GRASP is reported in column 12, 13 and 14 of Tables 1–3 respectively. Min–Min, SA and GRASP gave better schedules for the semiconsistent and consistent test cases. The percentage improvement of MVNS over Min–Min, SA and GRASP is found to be 23.5%, 82.3% and 83.6% respectively, by considering 72 test cases of various configurations.

Fig. 4 shows the average improvement of MVNS over other heuristic algorithms. It is revealed from Fig. 4 that the percentage improvement of MVNS is gradually increased when the problem dimension grows for SA and GRASP. Even though MVNS has better improvement over Min–Min algorithm, the percentage of improvement is gradually decreased for increasing problem dimension.

6.1.2. Speed of convergence

Fig. 5 illustrates the performance of MVNS, GRASP and SA algorithms during the search process, for the test case of Braun's semi-consistent, low job and low machine heterogeneity model with the configuration of 1024×32. It is found that the MVNS algorithm converges faster than the considered multi-objective algorithms with the exploration of shorter schedule.

6.1.3. Performance assessment

The comparison of two sets of non-dominated solutions obtained through two multi-objective optimization algorithms is important. In the literature, many performance assessment metrics for the multi-objective algorithms have been proposed [46–50]. This work makes use of the hyper volume difference indicator I_H^- for the performance assessment.

The reference set, R had been constructed by merging all of the archival non-dominated solutions found by each of the algorithms for a given configuration across 50 runs [51]. Then the hyper volume difference indicator I_H^- had been used to measure the differences between non-dominated fronts generated by the algorithms and the reference set R [51,52]. The objective values are normalized to find the hyper volume difference indicator [53]. I_H^- measures the portion of the objective space that is dominated by R. The lower the value of I_H^-, the better the algorithm performs [51].

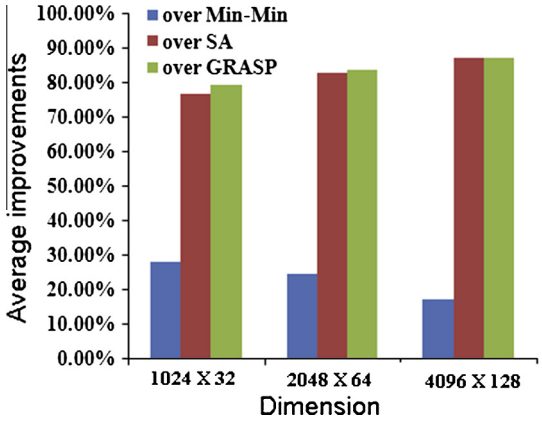

Figure 4 Average improvements of MVNS over other heuristics.

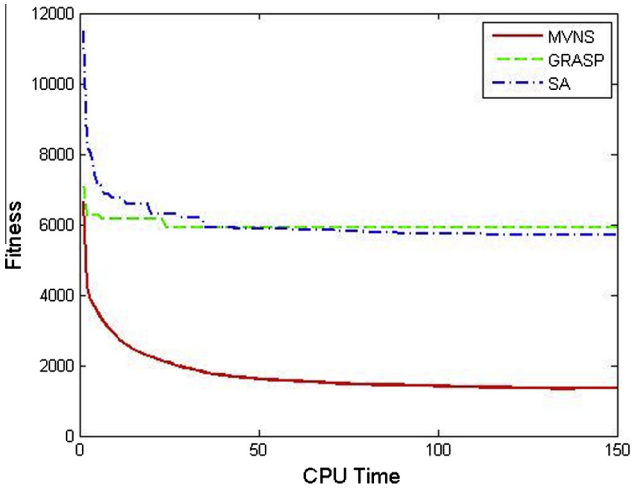

Figure 5 Convergence of MVNS, GRASP and SA.

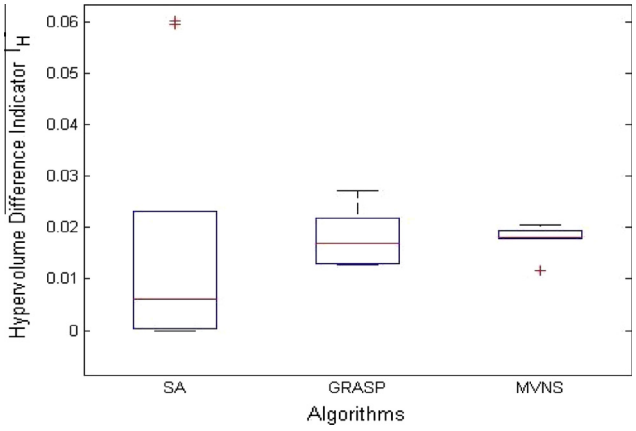

Figure 6 I_H^- measure of three algorithms for 1024×32.

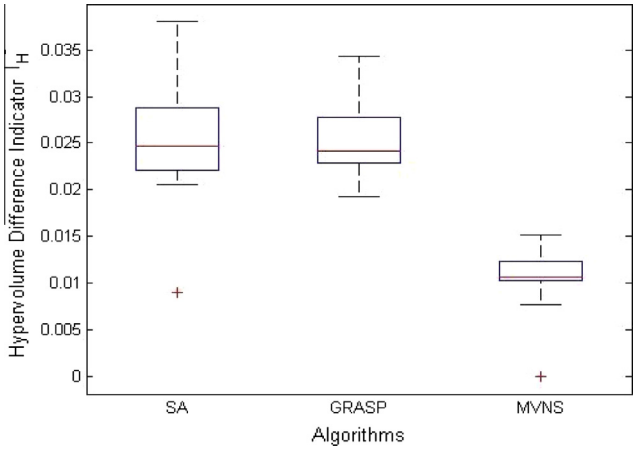

Figure 7 I_H^- measure of three algorithms for 2048×64.

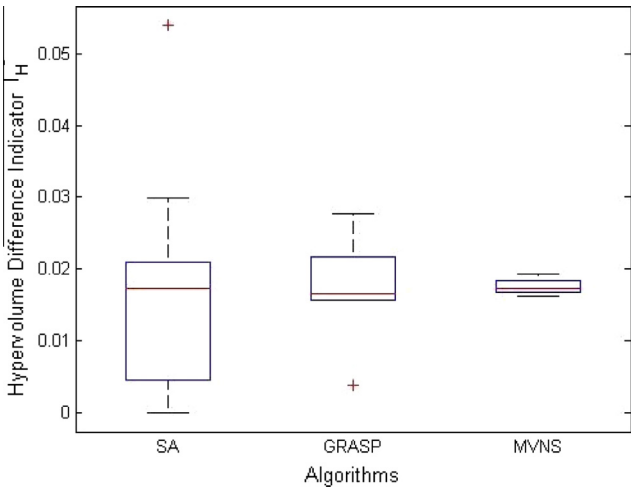

Figure 8 I_H^- measure of three algorithms for 4096×128.

The performance assessment plots had been drawn for the Ali's consistent, high job and high machine heterogeneity model, Braun's inconsistent, low job and high machine heterogeneity model, and Braun's semi-consistent, high job and high machine heterogeneity model with the configuration of 1024×32, 4096×128 and 4096×128 respectively.

Box plots for different configurations clearly prove that MVNS algorithm is better than GRASP and SA (Figs. 6–8). From the simulation result of MVNS algorithm in solving grid job scheduling problems, it is seen that the performance of MVNS algorithm is much better than other optimization techniques mentioned in this study.

7. Conclusions

Grid computing has emerged as one of the hot research areas in the field of computer networking. Scheduling, which decides how to distribute tasks to resources, is one of the most important issues. This paper presents the VNS algorithm with novel local search for grid job scheduling problem to minimize makespan and flowtime. Extensive computational experiments have been devised to study the performance of the proposed algorithm. The performance of MVNS was evaluated with other optimization algorithms, for a large variety of test cases, and with the consideration of the heterogeneous environment of different configurations. The results of MVNS are better for most of the instances. The computational results demonstrate the superiority of the proposed MVNS in solving the grid job scheduling problem and its computational efficiency. In future work, VNS algorithm for multi-objective complex scheduling problems and workflow model of grid scheduling problems will be developed.

References

[1] Foster I, Kesselman C, Tuecke. The anatomy of the grid: enabling scalable virtual organizations. Int J Supercomput Appl 2001;15:200–20.
[2] Luna F, Nebro AJ, Alba E. Observations in using grid-enabled technologies for solving multi-objective optimization problems. Parallel Comput 2006;32:377–93.
[3] Talbi EG. Parallel combinatorial optimization. USA: John Wiley & Sons; 2006.
[4] Talbi EG, Zomaya A. Grids for bioinformatics and computational biology. USA: John Wiley & Sons; 2007.
[5] Abraham A, Liu H, Grosan C, Xhafa F. Nature inspired metaheuristics for gird scheduling: single and multi-objective optimization approaches. In: Xhafa F, Abraham A, editors. Metaheuristics for scheduling in distributed computing environments, vol. 146. Berlin (Germany): Springer; 2008. p. 247–72.
[6] Ibarra OH, Ki CE. Heuristic algorithms for scheduling independent tasks on nonidentical processors. J ACM 1977;24:280–9.
[7] Brimberg J, Hansen P, Lih KW, Mladenovíc N, Breton M. An oil pipeline design problem. Oper Res 2003;51:228–39.
[8] Audet C, Brimberg J, Hansen P, Mladenovíc N. Pooling problem: alternate formulation and solution methods. Manage Sci 2004;50:761–76.
[9] Costa MC, Monclar FR, Zrikem M. Variable neighborhood decomposition search for the optimization of power plant cable layout. J Intell Manuf 2005;13:353–65.
[10] Meric L, Pesant G, Pierre S. Variable neighborhood search for optical routing in networks using latin routers. Ann Télécommun/Ann Telecommun 2004;59:261–86.
[11] Loudni S, Boizumault P, David P. On-line resources allocation for ATM networks with rerouting. Comput Oper Res 2006;33:2891–917.
[12] Hansen P, Mladenović N, Moreno Pérez JA. Variable neighborhood search: methods and applications. 4OR Quart J Oper Res 2008;6:319–60.
[13] Eshaghian M. Heterogeneous computing. Norwood: Artech House; 1996.
[14] Freund R, Sundaram V, Gottlieb A, Hwang K, Sahni S. Special issue on heterogeneous processing. J Parallel Distrib Comput 1994;21(3):255–6.
[15] Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of grid resource management systems for distributed computing. Software-Pract Exper 2002;32:135–64.
[16] Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distr Com 2001;61(6):810–37.
[17] Fidanova S. Simulated annealing for grid scheduling problem. In: Proceedings of IEEEJVA international symposium on modern computing. Bulgaria: IEEE Computer Society; 2006. p. 41–5.
[18] Chang R, Chang J, Lin P. An ANT algorithm for balanced job scheduling in grids. Future Gener Comput Syst 2009;25:20–7.
[19] Liu H, Abraham A, Hassanien AE. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener Comput Syst 2010;26:1336–43.
[20] Selvi S, Manimegalai D, Suruliandi A. Efficient job scheduling on computational grid with differential evolution algorithm. Int J Comput Theory Eng 2011;3:277–81.
[21] Nesmachnow S, Alba E, Cancela H. Scheduling in heterogeneous computing and grid environments using a parallel CHC evolutionary algorithm. Comput Intell 2012;28:131–55.
[22] Xhafa F, Carretero J, Dorronsora B, Alba E. Tabu search algorithm for scheduling independent jobs in computational grids. Comput Inform J 2009;28:237–49.
[23] Dueck G, Scheuer T. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 1990;90:161–75.
[24] Wen Y, Xu H, Yang J. A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. Inform Sci 2011;181:567–81.
[25] Xhafa F. A hybrid evolutionary heuristic for job scheduling on computational grids. Stud Comput Intell 2007;75:269–311.

[26] Abraham A, Liu H, Zhao M. Particle swarm scheduling for workflow applications in distributed computing environments. Stud Comput Intell 2008;128:327–42.
[27] Lusa A, Potts CN. A variable neighbourhood search algorithm for the constrained task allocation problem. J Oper Res Soc 2008;59:812–22.
[28] Moghaddam K, Khodadadi F, Entezari-Maleki R. A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment. Proc Eng 2014;29:3808–14, International workshop on information and electronics engineering.
[29] Xhafa F, Alba E, Dorronsoro B. Efficient batch job scheduling in grids using cellular memetic algorithms. J Math Model Alg 2008;7(2):217–36.
[30] Jacob W, Quinte A, Stucky KU, Sub W. Fastmulti-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm. In: Parallel problem solving from nature. Lecture notes in computer science, vol. 5199. Berlin: Springer; 2008. p. 1031–40.
[31] Xu J, Lam AYS, Li VOK. Chemical reaction optimization for task scheduling in grid computing. IEEE Trans Parallel Distr Syst 2011;22:1624–31.
[32] Schopf J. Ten actions when super scheduling, document of scheduling working group. Global Grid Forum; 2001. <http://www.ggf.org/documents/GFD.4.pdf>.
[33] Mateescu G. Quality of service on the grid via metascheduling with resource co-scheduling and co-reservation. Int J High Perform Comput Appl 2003;17:209–18.
[34] Khokhar AA, Prasanna VK, Shaaban ME, Wang CL. Heterogeneous computing: challenges and opportunities. IEEE Comput 1993;26:18–27.
[35] Siegel HJ, Dietz HG, Antonio JK. Software support for heterogeneous computing. ACM Comput Surv 1996;28:237–9.
[36] Cooper K, Dasgupta A, Kennedy K, Koelbel C, Mandal A, Marin G, et al. New grid scheduling and rescheduling methods in GrADS project. In: Proceeding of the 18th international parallel and distributed processing symposium (IPDPS '04). Santa Fe (New Mexico, USA); April 2004. p. 199–206.
[37] Dong F, Akl Selim G. Scheduling algorithms for grid computing: state of the art and open problems. Technical report 2006-504. Ontario, Kingston: Queen's University, School of Computing <ftp.qucis.queensu.ca/TechReports/Reports/2006-504.pdf>.
[38] Mladenović N, Hansen P. Variable neighborhood search. Comput Oper Res 1997;24:1097–100.
[39] Mladenović N, Hansen P. An introduction to variable neighborhood search. In: MetaHeuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer Academic; 1999. p. 449–67.
[40] Mladenović N, Hansen P. Variable neighborhood search: principles and applications. Euro J Oper Res 2001;130:449–67.
[41] Hansen P, Mladenović N. An introduction to variable neighborhood search. Handbook of metaheuristics. Amsterdam: Kluwer; 2003 [chapter 6].
[42] Hansen P, Mladenović N, Moreno Pé rez J. Developments of variable neighborhoodsearch. Ann Oper Res 2010;175(1):367–407.
[43] Alba E, Luque G. A new local search algorithm for the DNA fragment assembly problem. In: Proceedings of 7th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol. 4446. Springer; 2007. p. 1–12.
[44] Nesmachnow S, Cancela H, Alba E. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Appl Soft Comput 2012;12:626–39.
[45] Ali S, Siegel H, Maheswaran M, Hensgen D. Task execution time modelling for heterogeneous computing systems. In: Proceedings of the 9th heterogeneous computing workshop. Washington (USA): IEEE Press; 2000. p. 185.
[46] Deb K, Jain S. Running performance metrics for evolutionary multi-objective optimization. Technical report; 2002. doi: 10.1.1.9.159.
[47] Fonseca CM, Knowles JD, Thiele L, Zitzler E. A tutorial on the performance assessment of stochastic multiobjective optimizers. In: Third international conference on evolutionary multi-criterion optimization (EMO), vol. 216; 2005.
[48] Hansen MP, Jaszkiewicz A. Evaluating the quality of approximations to the non-dominated set. IMM, Department of Mathematical Modelling, Technical University of Denmark; 1998
[49] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evolut Comput 2000;8(2):173–95.
[50] Rego MF, Souza MJF, Coelho IM, Arroyo JEC. Multi-objective algorithms for the single machine scheduling problem with sequence-dependent family setups. In: The online conference on soft computing in industrial applications anywhere on earth; 2012.
[51] Talukder AKMKA, Kirley M, Buyya R. Multiobjective differential evolution for scheduling workflow applications on global Grids. Concurr Comp: Pract Exper 2009;21(13):1742–56.
[52] Huband S, Hingston P, Barone L, While L. A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans Evolut Comput 2006;10(5):477–506.
[53] Huang VL, Qin AK, Deb K, Zitzler E, Suganthan PN, Liang JJ, et al. Problem definitions for performance assessment on multi-objective optimization algorithms. Technical report. Nanyang Technological University, Singapore; 2007.