Arab J Sci Eng (2015) 40:817–844
DOI 10.1007/s13369-014-1558-9

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Task Scheduling Using Two-Phase Variable Neighborhood Search Algorithm on Heterogeneous Computing and Grid Environments

S. Selvi · D. Manimegalai

Received: 6 July 2014 / Accepted: 30 November 2014 / Published online: 24 January 2015

Abstract Grid computing solves high-performance and high-throughput computing problems through sharing nodes ranging from personal computers to supercomputers 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 Two-Phase Variable Neighborhood Search (TPVNS) algorithm for scheduling independent jobs on computational grid is carried out. The proposed algorithm consists of two modules with General Variable Neighborhood Search and Basic Variable Neighborhood Search algorithms in order to find a good mapping of grid jobs with grid nodes. The performance of the proposed algorithm has been evaluated with deterministic heuristic and evolutionary algorithms. Simulation results show that TPVNS algorithm generally performs better than the existing methods.

Keywords Grid computing · Job scheduling · Variable Neighborhood Search · Makespan

1 Introduction

There are various computation and data-intensive problems in science and industry that require weeks or months of computation to solve. Scientists involved in these types of problems need a computing environment that delivers large amounts of computational power over a long period of time. Such an environment is called a high-throughput computing (HTC) environment [1,2]. In HTC systems, there are mostly numbers of independent and sequential tasks which can be individually scheduled on many different computing resources across multiple administrative domains. HTC system can achieve this using grid computing technologies and techniques [3].

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data and storage or network nodes across dynamic and geographically dispersed organization [4]. Users can share grid nodes by submitting computing tasks to grid system. Nodes can be computers, storage space, instruments, software applications, and data. All nodes are connected through the Internet and a middleware layer that provide basic services for security, monitoring, and node management. The nodes of computational grid are dynamic, and they belong to different administrative domains. The participation of nodes may be active or inactive within the grid. This makes it impossible for anyone to manually assign jobs to computing nodes in grids. Therefore, grid job scheduling is one of the challenging issues in grid computing.

In order to achieve the HTC through grid environments, the overall response time to all the tasks in a relatively long period of time should be minimized. Therefore, the grid scheduler (GS) could schedule the submitted tasks on appropriate grid resources, considering the makespan of the environment. The throughput of the environment is increased by minimizing the total makespan of a grid environment [5,6]. To make effective use of the tremendous capabilities of the computational resources in grid environments and to minimize the makespan of the grids, efficient task scheduling algorithms are required. Scheduling problem in heteroge-

S. Selvi (✉)
Department of Electronics and Communication Engineering, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur 628215, Tamilnadu, India
e-mail: mathini31@yahoo.co.in

D. Manimegalai
Department of Information Technology, National Engineering College, Kovilpatti 628503, Tamilnadu, India
e-mail: megalai_nec@yahoo.co.in

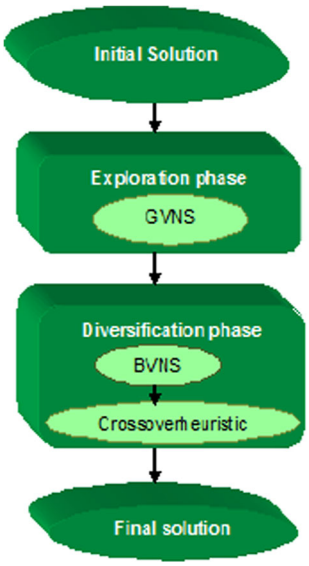

Fig. 1 Block diagram of TPVNS-based grid job scheduling algorithm

neous environments had been shown to be NP complete [7]. Hence, there are many research efforts aiming at job scheduling on the grid. The use of heuristics is the de facto approach in order to cope in practice with its difficulty.

This work presents a thorough experimental exploration of Two-Phase Variable Neighborhood Search algorithm implemented to execute in sequential mode, with problem-specific neighborhood structures to solve the grid job scheduling problem in order to reduce the makespan. The block diagram of the proposed algorithm is shown in Fig. 1.

The first module of the proposed algorithm is the exploration phase, which consists of General Variable Neighborhood Search (GVNS) algorithm. GVNS mainly focuses on the intensification of search process. The second phase is the diversification module, in which a new concept coined as crossover heuristic algorithm has been proposed. This accelerates the speed of the search process along with Basic Variable Neighborhood Search (BVNS) algorithm. The second phase is also helpful to identify new better solution and also to avoid the grid job scheduling algorithm to trap into the local minima. Extensive computational experiments were carried out to select an effective neighborhood order for the proposed Two-Phase Variable Neighborhood Search (TPVNS). Efficient numerical results are reported in the experimental analysis performed on a set of 124 well-known and large heterogeneous computing scheduling problem instances [8]. The comparative study shows that the proposed TPVNS is able to achieve high problem efficiency, outperforming the results of existing methods.

Variable Neighborhood Search (VNS) 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 demonstrated good performance on industrial applications such as design of an offshore pipeline network [9] and the pooling problem [10]. It has also been applied to real-world optimization problems, including optimization of a power plant cable layout [11], optical routing [12], and online nodes allocation problem for ATM networks [13]. Applications of VNS are diverse which include the areas such as location problems, data mining problems, graph problems, mixed integer problems, scheduling problems, vehicle routing problems, and biosciences and chemistry problems too [14,15].

2 Related Works

Some of the job scheduling algorithms are nature-inspired, e.g., Genetic Algorithm (GA) [16–19], cellular memetic algorithm [20], Simulated Annealing (SA) [21], ant colony optimization [22,23], Particle Swarm Optimization (PSO) [24], Differential Evolution (DE) [25], parallel Cross-generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation (pCHC) [8]. There are also non-nature-inspired metaheuristics, such as Tabu Search (TS) [26,27], Threshold Accepting (TA) [28], Chemical Reaction Optimization (CRO) [29], VNS algorithm [30].

Krauter et al. [31] provided a useful survey on grid node 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. [5] studied the comparison of the performance of batch queuing heuristics, TS, GA, and SA to minimize the makespan. The results revealed that GA achieved the best results compared with batch queuing heuristics. Xhafa [32] 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 combination of MA with other local search algorithms. Abraham et al. [33] 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.

Many recent articles have proposed novel concepts for solving the scheduling problem of heterogeneous computing and grid environments with various scheduling objectives, namely makespan [34–38], makespan and flowtime [39–41], energy consumption and makespan [42–46], makespan, flowtime and energy consumption [47,48], budget constraints and makespan [49–51], system reliability, system cost, deadline, quality of service constraints and redundancy [52,53], and load balancing factor [54,55].

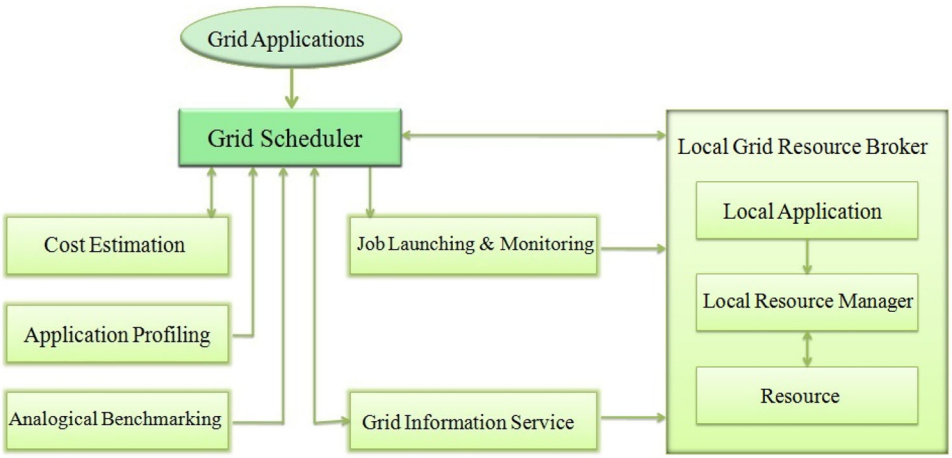

Fig. 2 A logical grid scheduling architecture

Neural- and fuzzy-based multi-criteria decision making scheduler and scheduler with novel resource provisioning policy have been proposed to enable efficient resource allocation and data transfer [56–60]. Navin et al. [61] proposed a new framework, namely Expert Grid, which is used to find, exploit, share, and manage the skills and knowledge of human resource, and which considers the optimal trade-off between the human resource and job demands. Lusa et al. [62] proposed the VNS algorithm for the constrained task allocation problem and compared the performance of the proposed algorithm with the other local search procedures. Kardani-Moghaddam et al. [63] presented a hybrid GA and VNS to reduce overall cost of task executions in grid environment.

The VNS algorithm has received relatively little attention in solving the grid job scheduling problem. It is known that VNS has been used in hybridization with other algorithms for such problems. There are no other antecedents on applying explicit VNS to solve the heterogeneous computing scheduling problems. So the approach presented here is a novel approach in this line of research to solve using VNS alone. In addition, none of the sequential execution algorithms outperformed the recently published results of parallel algorithm [8] for the de facto standard problems by Braun et al. [5]. This paves the path to contribute in these lines of research by studying sequential execution algorithms, able to deal with large-size scheduling problem instances by using innovative concepts embedded with sequential execution algorithms.

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 [4]. A grid scheduler (GS) receives applications from grid users, selects feasible nodes for these applications according to 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.

The grid scheduling process can be generalized into three stages, namely node discovering and filtering, node selecting and scheduling according to certain objectives, and job submission [64]. Figure 2 depicts a model of grid scheduling system. Grid scheduler is referred as meta-scheduler in the literature [65].

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 [66,67]. Cost estimation module computes the cost of candidate schedules. On the basis of knowledge from AP, AB, and cost estimation module, from which 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 [68].

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 [69] 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 preemption. 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. 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 [8].

4 Scheduling Problem Formulation

The problem is formulated based on the "Expected Time to Compute" (*ETC*) model [5]. 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 $n \gg m$) heterogeneous 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, we start with the following assumptions [29].

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

4.1 Mathematical Model

The standard three-field notation of Graham et al. [70] for scheduling problem is represented as $\alpha|\beta|\gamma$, where α represents the machine environment, β describes about the job characteristics, and γ refers to the optimality criterion chosen. The problem of this paper is denoted as $R||C_{max}$. Here, R represents the unrelated parallel machines scheduling problem. The *null* value of β denotes the absence of various characteristics of the job such as preemption, precedence relation, resource constraints, constant upper bound on machine allotted to the particular job, upper and lower bound on processing time. C_{max} denotes the makespan. Makespan is the completion time of the last finished task.

Based on the specifications of the nodes and tasks, meta-scheduler computes $n \times m$ matrix $ETC(ETC : J \times G \rightarrow R^+)$. The $R||C_{max}$ 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 goal of the grid job scheduling problem is to find an assignment of tasks to nodes (a function $f: J^n \rightarrow G^m$) which minimizes the makespan. The objective function can be expressed as follows:

$$\text{Minimize} \quad f(x) = max\left\{\sum_{[i/x_i=j]} ETC_{ij}\right\} \tag{1}$$

$$\text{s.t.} \quad x = \{x_1, x_2, \ldots, x_n\}, \quad \forall x_i \in [1, m], \forall i \in [1, n], \quad \forall j \in [1, m] \tag{2}$$

$$ETC_{ij} > 0, \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m \tag{3}$$

$$x_i^{(U)} = m, \quad i = 1, 2, \ldots, n \tag{4}$$

$$x_i^{(L)} = 1, \quad i = 1, 2, \ldots, n \tag{5}$$

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

$$C_j = \sum_{[i/x_i = j]} ETC_{ij}, \quad [i/x_i = j] \text{ represents the tasks assigned to node } j \tag{7}$$

In this model, the objective function (1) minimizes the makespan. Constraint (2) denotes a vector composed of n objective function parameters. Constraint (3) ensures that all entries of $n \times mETC$ matrix are positive. Constraints (4) and (5) define the upper and lower boundary constraints of the objective function parameters, respectively. Constraint (6) defines the upper and lower boundary constraints of the variable x_i. Constraint (7) calculates the completion time of node j, which is defined as the time required for node j to complete all its assigned tasks.

5 Variable Neighborhood Search 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 structure 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 Mladenovic' and Hansen [71]. Its advanced principles for solving combinatorial optimization problems and applications were further introduced in [72–74] and recently in [75].

VNS uses a finite set of preselected 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 $x \in X$, called as a local minimum, such that there exists no solution $x' \in N_k(x) \subseteq X$ 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. Usually the steepest direction of descent, also referred to as the best improvement, is used. The steps of the best improvement are given in Algorithm 1.

After the local search, a change in the neighborhood structure is performed. The generic form of the neighborhood change function is given in Algorithm 2. 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.

Algorithm 1. BestImprovement

Input: x (initial solution)
Output: x (local minimum)
1 ***repeat***
2 $x' \leftarrow x$
3 $x \longleftarrow \arg\ min_{y \in N_k(x)} f(y)$
4 ***until*** $f(x') > f(x)$

The VNS can be summarized as in Algorithm 3. VNS uses two parameters: β, which is the maximum number of iterations allowed as the stopping condition, and k_{max}, which is the number of neighborhood structures used. Step 4 of Algorithm 3, which is called shaking, randomly chooses a solution x' from the kth neighborhood of the incumbent solution x. After improving this solution via the BestImprovement local search (Algorithm 1), a neighborhood change is employed with the function NeighborhoodChange (Algorithm 2).

Algorithm 2. NeighborhoodChange

Input: x,x',k
Output: x,k
1 ***if*** $f(x') < f(x)$ ***then***
2 $x \leftarrow x'$
3 $k \leftarrow 1$
4 ***else***
5 $k \leftarrow k + 1$
6 ***endif***

If we eliminate the randomness in VNS, then the Variable Neighborhood Descent (VND) is obtained (Algorithm 4). While this deterministic variant of VNS can be used as it is, it might be useful as a local search within a VNS. In the latter case, one can obtain a better solution at the end of the local search since VND itself uses more than one neighborhood structure. Hence, the chances to reach a global solution are larger when using VND rather than a single neighborhood structure [75].

As a variant of the VNS, if the local search step of the VNS is replaced by VND, then we obtain the GVNS [76].

Algorithm 3. Basic Variable Neighborhood Search

Input: x, k_{max}, β
Output: x
1 ***for*** $iter\_VNS = 1$ *to* β ***do***
2 $k \leftarrow 1$
3 ***repeat***
4 $x' \leftarrow shake(x,k)$
5 $x'' \leftarrow BestImprovement(x')$ /* Local search */
6 $Neighborhoodchange(x,x'',k)$
7 ***until*** $k = k_{max}$
8 ***endfor***

Algorithm 4. Variable Neighborhood Descent

Input: x, k_{max}
Output: x
1 $k \leftarrow 1$
2 ***repeat***
3 $x' \leftarrow \arg\ min_{y \in N_k(x)} f(y)$
4 $Neighborhoodchange(x,x',k)$
5 ***until*** $k = k_{max}$

Steps of the GVNS are given in Algorithm 5. GVNS uses one additional parameter other than β and k_{max}, that is k'_{max}, the number of neighborhoods used in the inner VND loop.

The neighborhood change function described in Algorithm 2 consists of general procedure of fitness evaluation. But VNS-based job scheduling algorithm consists of GVNS and BVNS modules, in which neighborhood change function evaluates the fitness of the solution using Algorithms A.1 and A.2, respectively.

6 Implementation of Two-Phase VNS 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.

6.1 Solution Representation

The solution is represented as an array of length equal to the number of jobs. The value corresponding to each position *i* 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. 3. 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.

6.2 Initial Solution Generation

Numerous methods have been proposed to generate the initial solution when applying metaheuristics to the scheduling problem in the heterogeneous environment [32,33,77]. The deterministic heuristic *Min–Min* algorithm has been used as a method to generate the initial solution (Algorithm A.3). This algorithm leads to more balanced schedules and generally finds smaller makespan values than other heuristics, since more tasks are expected to be assigned to the nodes that can complete them the earliest. As the *Min–Min* algorithm provides a good starting solution to the grid job scheduling algorithm, the TPVNS algorithm converges to a desired solution faster than when using the random initial solution.

6.3 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 details of the neighborhood structures experimented in this paper are given in "Appendix A" (Algorithms A.4, A.5,

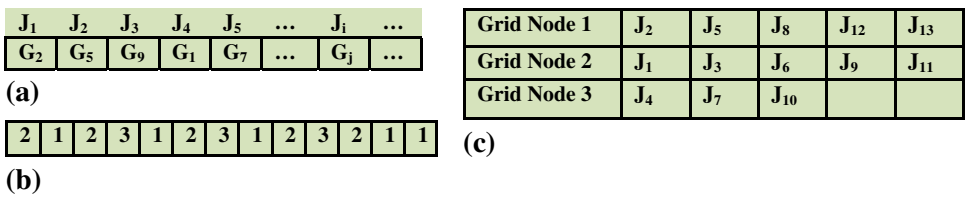

Fig. 3 **a** Solution representation, **b** solution for the problem of 13 jobs and 3 grid nodes, **c** mapping of jobs with grid nodes for the solution given in (**b**)

A.6, A.7, A.8, and A.9). The defined neighborhood structures and corresponding moves associated with them are explained in detail below.

6.3.1 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.

6.3.2 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.

6.3.3 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.

6.3.4 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.

6.3.5 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.

6.3.6 Problem Aware Local Search (PALS)

Basic concept of this neighborhood structure has been used in the literature for the DNA fragment assembly problem [78] and the heterogeneous computing scheduling problem [79]. Working on a given schedule x, this neighborhood 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 line 3 and 5 of Algorithm A.9.). The randomness introduced in the parameters *endheavy* and *endres* makes this neighborhood to differ from the concept existing in the literature.

To illustrate, a small-scale job scheduling problem involving 3 nodes and 13 jobs is considered. The node speeds are 4, 3, 2 cycles/s, and the job lengths of 13 jobs are 6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, and 60 cycles, respectively. Consider the initial solution with makespan 115, which is represented in Fig. 4a. 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 makespan of the solution as 105 (Fig. 4b). 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 makespan of the current solution becomes 102, which is illustrated in Fig. 4c. 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), but maintains the makespan of current solution (Fig. 4d). According to the Weightedmakespan–InsertionMove, the job J_{13} from the job list of G_3 (considered as *Hr*) is assigned to the node G_1 (considered as *Lr*). This neighborhood minimizes the makespan of current solution as 72 (Fig. 4e). 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. 4f). Hence, the final solution has the makespan 46, which is the optimal result for the example problem.

6.4 Proposed Grid Job Scheduling Algorithm

This section describes the proposed VNS-based job scheduling algorithm. The pseudo code is detailed in Algorithm 6. The first phase is the generation of initial solution which acts as the seed for the job scheduling algorithm. The deterministic heuristic, Min–Min algorithm helps the job scheduling algorithm by providing better starting solution to explore into the solution space, thereby increasing the speed of the search process.

The proposed algorithm consists of two modules, namely exploration and diversification. The exploration phase makes use of GVNS algorithm, which concentrates on the minimization of makespan of the solution. GVNS algorithm has N_k neighborhood structures, ($k = 1$ to k_{max}) in the shaking

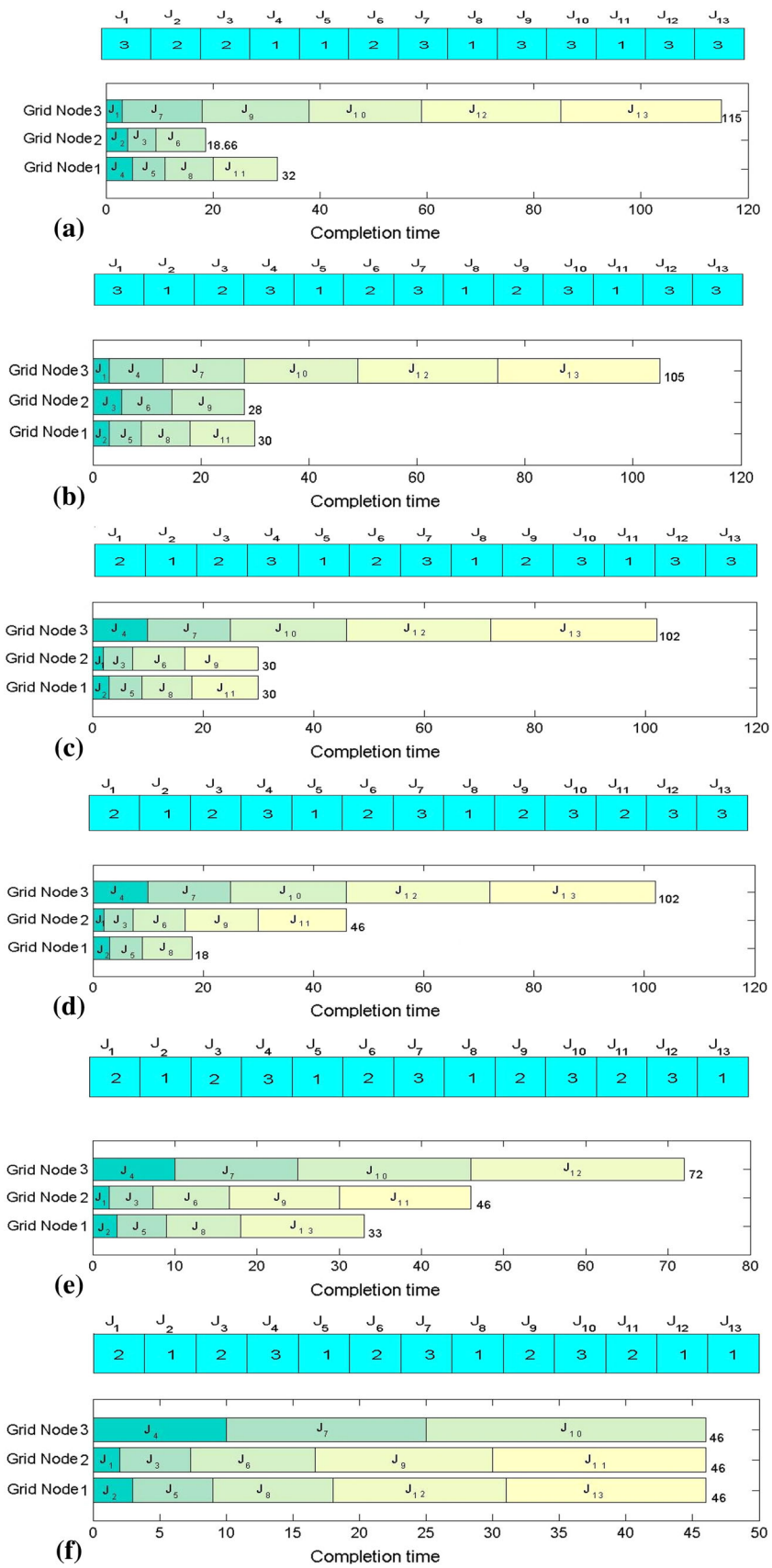

Fig. 4 Explanation of different neighborhood structures used in the proposed TPVNS: **a** Initial solution, **b** SwapMove, **c** Makespan–InsertionMove, **d** InsertionMove, **e** Weightedmakespan–InsertionMove, and **f** BestInsertionMove

Algorithm 5. General Variable Neighborhood Search

Input: $x, k_{max}, k'_{max}, \alpha$

Output: x

1 ***for*** $iter\_GVNS = 1$ *to* α ***do***
2 $k \leftarrow 1$
3 ***repeat***
4 $x' \leftarrow shake(x,k)$
5 $x'' \leftarrow VND(x', k'_{max})$
6 $Neighborhoodchange(x, x'', k)$
7 ***until*** $k = k_{max}$
8 ***endfor***

Algorithm 6. VNS based grid job scheduling algorithm

Input: $ETC[\,][\,], k_{max}, k'_{max}, k''_{max}, t_{max}, t\_initial$, $threshold$, $\alpha, \beta, \gamma, \phi$, $PALS\_maxiter$, $bit\_difference$
Output: x

1 $Generation \leftarrow 0$; $\phi_{tot} \leftarrow cputime + t\_initial$; $\gamma_{tot} \leftarrow 1$; $make\_span[\,] \leftarrow 0$;
2 $x \leftarrow Min\text{-}Min\ (ETC[\,][\,])$ /* Algorithm A.3 */
3 ***repeat***
4 $GVNS(x, k_{max}, k'_{max}, \alpha)$ /* Algorithm 5 */
5 $[localmakespan[\,], fitnes] \leftarrow f_2(x, ETC[\,][\,])$ /* Algorithm A.2 */
6 $BVNS(x, k''_{max}, \beta)$ /* Algorithm 3 */
7 $[localmakespan[\,], fitnes] \leftarrow f_1(x, ETC[\,][\,])$ /* Algorithm A.1 */
8 $[x\,,\ \gamma_{tot}] \leftarrow Crossoverheuristic(Generation, threshold, bit\_difference, x, \gamma, \phi, \phi_{tot}, \gamma_{tot})$
9 $Generation \leftarrow Generation + 1$
10 $solution[Generation] \leftarrow x$
11 $[localmakespan[\,], fitnes] \leftarrow f_1(x, ETC[\,][\,])$
12 $make\_span(Generation) \leftarrow fitnes$
13 ***if*** $(Generation > 3)$
14 ***if*** $(make\_span[Generation] == make\_span[Generation - 1])$
15 $\gamma_{tot} \leftarrow \gamma_{tot} + 1$
16 ***else***
17 $\gamma_{tot} \leftarrow 1$
18 ***endif***
19 ***endif***
20 $t \leftarrow cputime()$
21 ***until*** $t = t_{max}$

phase to create new solution from the current solution and $N_{k'}$ neighborhood structures, ($k' = 1$ to k'_{max}) in the VND phase to improve this new solution, which is framed based on Eq. (1). After α times execution of GVNS module, the makespan of the current solution will be improved by the diversification phase of the scheduling algorithm.

The diversification phase of the scheduling algorithm consists of two modules. The first module consists of BVNS algorithm, which has k''_{max} neighborhood structures, ($k'' = 1$ to k''_{max}) in the shaking phase with local search heuristic. The BVNS algorithm works on different strategy of evaluating the fitness of the solution as detailed in Algorithm A.2. Hence, the fitness of the current solution is calculated using Algorithm A.2, and then, the solution is submitted to the BVNS module. Algorithm A.2 is formulated based on Eq. (9), which considers the average flowtime and makespan to estimate the fitness of current solution. *Flowtime* is the total time consumed by all tasks [29]. Mathematically, we have

$$Flowtime = \sum_{j=1}^{m} \left(\sum C_j \right) \tag{8}$$

$$Fitness = a \times Makespan + (1 - a) \times \left(\frac{Flowtime}{m} \right) \tag{9}$$

where C_j is the completion time of grid node j to complete all its assigned tasks which is defined in Eq. (7), "a" is fixed as 0.75 [32], and m is the total number of grid nodes. This module of diversification phase is regularly called by the scheduling algorithm for β times, after every α times execution of exploration phase.

In order to further diversify the search, the second module is constructed, which evaluates the fitness of the solution using Algorithm A.1. Hence, before applying the schedule to the second module of diversification phase, the fitness of the current solution is modified by Algorithm A.1. Algorithm 7 describes the method of diversification procedure of second module adopted to improve the speed of the search process.

The second module is invoked regularly at every ϕ seconds during the execution of scheduling algorithm and also whenever there is no improvement in the makespan of the current solution for successive γ generations. Algorithm 7 considers the current solution and the randomly selected previous generation solution in order to reassign the grid nodes with jobs. The previous generation solution is selected by considering the concept that the previous should differ in at least 2 bit positions from the current solution. The solutions generated by Algorithm 7 are manipulated based on two conditions in order to update the current solution. The current solution is updated by the new one if the new solution has less makespan than the current or if the difference between the makespan of new and current is less than $threshold$.

The diversification phase of the scheduling algorithm may yield good solution or sometimes deteriorated solution. Even though this phase yields deteriorated solution, the reassignment of grid nodes with jobs done by the diversification phase of grid job scheduling algorithm helps the GVNS algorithm to explore new better solution.

7 Computational Experiments

When facing the heterogeneous computing scheduling problem, researchers have often used twelve instances proposed by Braun et al. [5], following the *ETC* performance estimation model by Ali et al. [80]. *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

Algorithm 7. Crossoverheuristic

Input : $Generation, threshold, bit\_difference, x, \gamma, \phi, \phi_{tot}, \gamma_{tot}$
Output: x, γ_{tot}

$x' \leftarrow x$
if $((\gamma_{tot} == \gamma) || (cputime( ) > \phi_{tot}))$ **then**
  if $(\gamma_{tot} \sim = \gamma)$ **then**
    $\phi_{tot} \leftarrow cputime + \phi$
  endif
  $g \leftarrow solution(randint(1, Generation))$
  Jobindex[ ] ← compare_bit_position(g, x') /* Return the index of bitposition where g differs from x' */
  while *(length(jobindex[])<bit_difference)* /*Repeat the procedure to maintain atleast 2 bit difference*/
    $g \leftarrow solution(randint(1, Generation))$
    Jobindex[ ] ← compare_bit_position(g, x')
  endwhile
  $J_1[\,] \leftarrow Jobindex(randint(1, length(jobindex[\,])))$
  $J_2[\,] \leftarrow Jobindex(randint(1, length(jobindex[\,])))$ /* $J_1 \neq J_2$ */
  $interchange(x'[J_1], g[J_1])$ /* Interchange the grid nodes */
  $interchange(x'[J_2], g[J_2])$
  $[localmakespan\_g, fitnes\_g] \leftarrow f_1(g, ETC[\,][\,])$
  $[localmakespan\_x', fitnes\_x'] \leftarrow f_1(x', ETC[\,][\,])$
  $[localmakespan\_x, fitnes\_x] \leftarrow f_1(x, ETC[\,][\,])$
  if $(fitnes\_x' < fitnes\_g)$ **then**
    if $(fitnes\_x' \leq fitnes\_x)$ **then**
      $x \leftarrow x'$
      elseif$\left(\left(\frac{fitnes\_x' - fitnes\_x}{fitnes\_x'}\right) < threshold\right)$
      $x \leftarrow x'$
      endif
    endif
  elseif$(fitnes\_g \leq fitnes\_x)$ **then**
    $x \leftarrow g$
    elseif$\left(\left(\frac{fitnes\_g - fitnes\_x}{fitnes\_g}\right) < threshold\right)$
      $x \leftarrow g$
      endif
    endif
  endif
  if$(\gamma_{tot} == \gamma)$
    $\gamma_{tot} = 1$
  endif

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, 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.

All instances from Braun et al. [5] are composed of 512 jobs and 16 machines, which is referred as the configuration 512×16. They are labeled as *u_x_yyzz.k* where *u* means uniform distribution (in the *ETC* matrix generation), *x* is the type of consistency (*c*—consistent, *i*—inconsistent, and *s* means semiconsistent), *yy* and *zz* indicate the job and machine heterogeneity (*hi*—high, and *lo*—low), and *k* is used to number instances of the same type. Several test suites were generated, but only the class 0 ($k = 0$) gained popularity. Nesmachnow et al. [8] 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. Each dimension has 24 test instances regarding all the heterogeneity and consistency combinations, twelve of them considering the parameterization values from Ali et al. [80], and twelve using the values from Braun et al. [5]. The instances are named following the previously presented convention: The names have the pattern *M.u_x_yyzz*, where the first letter (*M*) describes the heterogeneity model (*A* for Ali, and *B* for Braun). Liu et al. [24] used a test suite for grid job scheduling problem, which comprises 4 different configurations represented as (number of grid nodes, number of grid jobs), namely (3, 13), (8, 60), (5, 100), and (10, 50).

Braun et al. [5] carried out the experiment for 512×16 configuration with Pentium II 400 MHz processor (1 GB RAM) and reported the best result using GA with an average execution time of 60 s. Ritchie et al. [23] took around 3.5 h for hybrid ACO+TS algorithm to give the best result over GA with 1.6 GHz processor. Xhafa et al. [27] set 100 s for TS to provide the best result over hybrid ACO+TS algorithm with Pentium III 550 MHz processor (256 MB RAM). Nesmachnow et al. [8] fixed 90 s for pCHC to present best result over TS (512×16) and 120 s for large-scale problems. They used a cluster with four Dell PowerEdge servers with Quad-Core Xeon E5430 processors at 2.66 GHz, 8 GB RAM, using the CentOS Linux 5.2 operating system, connected with a Gigabit Ethernet LAN for experimentation. Liu et al. [24] experimented the PSO algorithm by setting $50 \times m \times n$ iterations (m is the number of grid nodes, n is the number of jobs), which took around 160–1585 s for different dimension.

This paper considers the test instances with dimension 512×16 [19], $1{,}024 \times 32$, $2{,}048 \times 64$, and $4{,}096 \times 128$ [8] and the test suite used by Liu et al. [24]. Also, it has been decided to generate 3 different configurations of *ETC* matrix considering the parameterization values from Braun et al. [5] with dimension 100×10, 300×10, and 512×16, and to develop and run Min–Min algorithm, greedy randomized adaptive search procedures (GRASP), and SA on the same platform in order to make a fair comparison. The additional test instances are named using the pattern *x_yyzz*, by following the previously presented convention. Thus, 124 different test instances are used to evaluate the performance of the proposed algorithm.

The VNS-based grid job scheduling algorithm was developed using MATLAB R2010a and run on an Intel(R) Core(TM) i5 2.67 GHz CPU with 4GB RAM. The evaluation of the fitness function usually requires larger computing time than the application of neighborhood operators. It is found that fitness evaluation of single solution consumes 2.63, 2.756, 2.812, and 5.827 ms for 512×16, $1{,}024 \times 32$, $2{,}048 \times 64$, and $4{,}096 \times 128$ dimension problems of Nesmachnow et al., respectively. Hence, the maximum running time of the algorithm is not set to uniform value for all configurations. The stopping condition t_{max} is set to 5, 17, 50, 75, 150, 300, and 700 s for 3×13, 100×10, 300×10, 512×16, $1{,}024 \times 32$, $2{,}048 \times 64$, and $4{,}096 \times 128$ dimension prob-

lems, respectively. In reality, the stopping criteria could be that the algorithm reaches a preset value $t_{max,}$ which is set to 50 s for small-sized test instances ($\leq 512 \times 16$ dimension), 150 s for medium-sized test instances ($512 \times 16 <$ dimension $\leq 2{,}048 \times 64$), and 300 s for large-sized test instances ($2{,}048 \times 64 <$ dimension $\leq 4{,}096 \times 128$).

7.1 Examining the Performance of the Algorithm Parameters

This section deals with the experimentation to make three algorithm-specific decisions. First, the order of execution of neighborhoods in the exploration phase and the selection of neighborhood in the diversification phase had been decided. Then, the role of diversification in the job scheduling algorithm had been emphasized. Next, the algorithm-specific parameters, namely α, β, γ, ϕ and *threshold* had been determined. For taking these decisions, the proposed algorithm was run for the test instances, namely A.u_c_hilo and B.u_s_lolo of $1{,}024 \times 32$ and $2{,}048 \times 64$ configuration obtained from the test suite proposed by Nesmachnow et al. [8]. Each experiment (for each instance) was repeated 50 times, and the best makespan was tabulated.

In order to establish the best order of the six neighborhood structures discussed in Sect. 6.3, for the GVNS and BVNS algorithm, the preliminary computational experiments were performed with the value of α, β, γ, ϕ and *threshold* fixed at 3, 6, 3, 15, and 0.0003, respectively. For this study, SwapMove and InsertionMove are used in the VND phase of GVNS. Makespan–InsertionMove and Problem Aware Local Search heuristic are used in the shaking and local search phase of BVNS algorithm, respectively, to carry out the study. The proposed algorithm uses 8 parameters, and their description is listed below:

- α and β are the maximum number of times of execution of GVNS and BVNS algorithm per single generation, respectively.
- γ denotes the maximum number of generations for which the algorithm yields the same value of makespan.
- At every ϕ seconds, the crossover heuristic is called during the execution of job scheduling algorithm. The crossover heuristic gives different direction for the GVNS phase to explore into the solution space; γ and ϕ are the parameters which decide the process of invoking the crossover heuristic.
- $t\_initial$ is the time (in seconds) at which the crossover heuristic is called initially after starting the execution of job scheduling algorithm.
- *PALS_maxiter* is the parameter which decides the maximum number of times of execution of the PALS heuristic (Algorithm A.10).
- $bit\_difference$ is the number of bits used in the swapping process of crossover heuristic.

The parameters $t\_initial$, *PALS_maxiter* and $bit\_difference$ are set to 10, 5, and 2, respectively.

7.1.1 Order of Neighborhood Structures

Different ordering of the proposed six neighborhoods with their subsets is listed in Table 1, where the numbers refer to the test case numbers corresponding to the neighborhood list. The best makespan obtained during the experimentation of each test case for the determination of the order of the neighborhood structures in the shaking phase of GVNS is given in Table 2. As seen from Table 2, three test cases gave better results compared to other cases which are highlighted in bold. The best result was obtained for the test case 44. This test case consists of 2 neighborhood structures, namely Problem Aware Local Search and SwapMove. With this ordering of neighborhood structures in the shaking phase, the experiment was again repeated to determine the order of neighborhood structures in the VND phase of GVNS algorithm. The results for the experimentation of different neighborhood structures in the VND phase of exploration module are given in Table 3. From Table 3, it is observed that the test case with the combination of Weightedmakespan–InsertionMove, BestInsertionMove, and InsertionMove neighborhoods in the VND phase of GVNS gave better mapping of jobs with grid nodes, which is highlighted in bold. Thus, k_{max} and k'_{max} are set to 2 and 3, respectively.

The first phase of diversification module of grid job scheduling algorithm consists of BVNS algorithm, in which neighborhood change function considers Algorithm A.2 for fitness evaluation. But the objective of the grid job scheduling algorithm is to find good solution, whose fitness is evaluated using Algorithm A.1. Hence, it is required to have very slight perturbation of incumbent solution, but not to have more intensification of search process in the solution space of the BVNS algorithm. Thus, k''_{max} is set to 1. Table 4 reports the experimental result of grid job scheduling algorithm for the selection of suitable neighborhood structure in the shaking phase of BVNS algorithm. Weightedmakespan–InsertionMove neighborhood (used for the initial study) gave the minimum makespan value which is highlighted in bold when compared to other neighborhood structures. Table 5 shows the best makespan value during the experimentation of different neighborhood structure used in the local search phase of the BVNS algorithm of the grid job scheduling algorithm. The usage of PALS heuristic reports the minimum makespan.

Table 1 Different combinations and orders of the shaking procedures for GVNS

Neighborhood structure test cases (1–44)							
1	Problem Aware Local Search InsertionMove Makespan–InsertionMove BestInsertionMove Weightedmakespan–InsertionMove	12	SwapMove InsertionMove Makespan–InsertionMove BestInsertionMove Weightedmakespan–InsertionMove	23	SwapMove InsertionMove Problem Aware Local Search BestInsertionMove Weightedmakespan–InsertionMove	34	SwapMove InsertionMove Problem Aware Local Search BestInsertionMove Makespan–InsertionMove
2	Weightedmakespan–InsertionMove Problem Aware Local Search Makespan–InsertionMove BestInsertionMove InsertionMove	13	Weightedmakespan–InsertionMove SwapMove Makespan–InsertionMove BestInsertionMove InsertionMove	24	Weightedmakespan–InsertionMove SwapMove Problem Aware Local Search BestInsertionMove InsertionMove	35	Makespan–InsertionMove SwapMove Problem Aware Local Search BestInsertionMove InsertionMove
3	InsertionMove Makespan–InsertionMove BestInsertionMove Weightedmakespan–InsertionMove Problem Aware Local Search	14	InsertionMove Makespan–InsertionMove BestInsertionMove Weightedmakespan–InsertionMove SwapMove	25	InsertionMove Problem Aware Local Search BestInsertionMove Weightedmakespan–InsertionMove SwapMove	36	InsertionMove Problem Aware Local Search BestInsertionMove Makespan–InsertionMove SwapMove
4	Makespan–InsertionMove InsertionMove BestInsertionMove Weightedmakespan–InsertionMove Problem Aware Local Search	15	Makespan–InsertionMove InsertionMove BestInsertionMove Weightedmakespan–InsertionMove SwapMove	26	Problem Aware Local Search InsertionMove BestInsertionMove Weightedmakespan–InsertionMove SwapMove	37	Problem Aware Local Search InsertionMove BestInsertionMove Makespan–InsertionMove SwapMove
5	BestInsertionMove InsertionMove Weightedmakespan–InsertionMove Problem Aware Local Search Makespan–InsertionMove	16	BestInsertionMove InsertionMove Weightedmakespan–InsertionMove SwapMove Makespan–InsertionMove	27	BestInsertionMove InsertionMove Weightedmakespan–InsertionMove SwapMove Problem Aware Local Search	38	BestInsertionMove InsertionMove Makespan–InsertionMove SwapMove Problem Aware Local Search
6	Weightedmakespan–InsertionMove Problem Aware Local Search InsertionMove Makespan–InsertionMove	17	Weightedmakespan–InsertionMove SwapMove InsertionMove Makespan–InsertionMove	28	Weightedmakespan–InsertionMove SwapMove InsertionMove Problem Aware Local Search	39	Makespan–InsertionMove SwapMove InsertionMove Problem Aware Local Search
7	InsertionMove Problem Aware Local Search Makespan–InsertionMove BestInsertionMove	18	SwapMove InsertionMove Makespan–InsertionMove BestInsertionMove	29	SwapMove InsertionMove Problem Aware Local Search BestInsertionMove	40	InsertionMove Problem Aware Local Search BestInsertionMove SwapMove
8	Problem Aware Local Search InsertionMove Makespan–InsertionMove	19	InsertionMove SwapMove Makespan–InsertionMove	30	InsertionMove SwapMove Problem Aware Local Search	41	SwapMove InsertionMove Problem Aware Local Search
9	InsertionMove Makespan–InsertionMove BestInsertionMove	20	InsertionMove BestInsertionMove Makespan–InsertionMove	31	InsertionMove Problem Aware Local Search BestInsertionMove	42	InsertionMove BestInsertionMove Problem Aware Local Search

Table 1 continued

Neighborhood structure test cases (1–44)							
10	InsertionMove Makespan–InsertionMove	21	BestInsertionMove Makespan–InsertionMove	32	InsertionMove Problem Aware Local Search	43	Problem Aware Local Search Weightedmakespan–InsertionMove
11	Problem Aware Local Search Makespan–InsertionMove	22	SwapMove Makespan–InsertionMove	33	SwapMove BestInsertionMove	44	Problem Aware Local Search SwapMove

Table 2 Neighborhood structure testing for the shaking phase of GVNS

Test case	Makespan		Test case	Makespan	
	B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)		B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
1	1,027.0	5,345,167.7	23	1,050.6	5,423,853.6
2	1,035.8	5,322,253.0	24	1,051.1	5,446,843.0
3	1,044.5	5,346,058.8	25	1,049.9	5,416,835.9
4	1,049.7	5,378,553.7	26	1,023.4	5,382,384.8
5	1,037.7	5,318,607.4	27	1,056.1	5,485,028.4
6	1,035.1	5,369,010.4	28	1,044.3	5,360,014.2
7	1,043.4	5,320,008.8	29	1,051.5	5,467,845.8
8	**1,022.7**	**5,291,502.2**	30	1,050.7	5,311,420.2
9	1,049.5	5,344,381.3	31	1,036.4	5,348,914.3
10	1,041.7	5,353,521.0	32	1,051.7	5,429,781.2
11	1,028.3	5,308,613.3	33	1,047.7	5,458,596.7
12	1,027.1	5,368,480.1	34	1,037.8	5,402,048.0
13	1,031.0	5,351,404.9	35	1,028.9	5,393,975.0
14	1,022.6	5,351,057.2	36	1,027.7	5,334,054.5
15	1,029.5	5,310,227.3	37	1,024.1	5,380,725.6
16	1,025.4	5,408,003.1	38	1,038.5	5,342,215.5
17	1,023.5	5,382,143.0	39	1,029.1	5,363,748.5
18	1,026.1	5,304,964.9	40	1,045.1	5,304,345.5
19	1,041.8	5,355,614.6	**41**	**1,023.0**	**5,303,312.5**
20	1,036.6	5,335,828.1	42	1,032.4	5,352,345.5
21	1,030.2	5,352,555.8	43	1,034.2	5,311,498.5
22	1,045.0	5,412,338.2	**44**	**1,021.6**	**5,269,210.5**

7.1.2 Parameter Setting

The proposed grid job scheduling algorithm consists of exploration and diversification phase. Grid job scheduling algorithm requires diversification procedure to lead way for exploring new better solution. But this should not spend much time in the diversification module. The balanced usage of exploration and diversification module is important for the job scheduling algorithm to produce good solution. The first phase of diversification process is an executed β time, which is called regularly after α times execution of exploration module. The parameters γ and ϕ are used in the second phase of the diversification process, which decide the frequency of calling the crossover heuristic. The second phase utilizes different strategy of diversification, which deteriorates the incumbent solution if the crossover heuristic fails to yield good solution. The parameter *threshold* used by the second phase has to be tuned properly, since *threshold* decides the level of degradation of better solution. Hence, experiments were conducted to determine suitable value for the algorithm-specific parameters, namely $\alpha, \beta, \gamma, \phi$, and *threshold*. Table 6 reports the makespan value for different combination of α and β. It is observed from Table 6 that the grid job scheduling algorithm is able to find better mapping of grid jobs with grid nodes for the combination of α and β fixed at 3 and 6 (used for the initial study), respectively. The makespan values obtained during the experimentation of grid job scheduling algorithm for different combination of γ, ϕ,

Table 3 Neighborhood structure testing for the VND

Test case	Order of neighborhood	Makespan	
		B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
1	SwapMove → InsertionMove	1,021.6	5,269,210.5
2	Weightedmakespan–InsertionMove →Makespan–InsertionMove	1,068.1	5,512,662.1
3	BestInsertionMove → Weightedmakespan–InsertionMove	1,052.4	5,486,890.8
4	Weightedmakespan–InsertionMove → BestInsertionMove	1,047.6	5,521,421.3
5	Makespan–InsertionMove→ PALS	1,036.0	5,384,861.1
6	Weightedmakespan–InsertionMove → InsertionMove	1,034.5	5,321,268.0
7	PALS → BestInsertionMove→ InsertionMove	1,027.4	5,388,025.2
8	InsertionMove→ Weightedmakespan–InsertionMove → Makespan–InsertionMove	1,031.7	5,361,900.8
9	BestInsertionMove → Weightedmakespan–InsertionMove→ InsertionMove	1,016.4	5,300,079.0
10	Makespan–InsertionMove →SwapMove →PALS	1,026.4	5,360,079.5
11	**Weightedmakespan–InsertionMove → BestInsertionMove→ InsertionMove**	**1,010.2**	**5,200,512.2**

Table 4 Neighborhood structure testing for BVNS

Type of neighborhood	Makespan	
	B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
SwapMove	1,021.6	5,229,200.0
Makespan–InsertionMove	1,010.2	5,200,512.2
InsertionMove	1,020.6	5,299,200.0
Weightedmakespan–InsertionMove	**1,001.5**	**5,186,352.2**
BestInsertionMove	1,042.0	5,378,337.9
PALS	1,017.8	5,318,608.4

Table 5 Neighborhood structure testing for the Local search of BVNS

Type of neighborhood	Makespan	
	B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
SwapMove	1,021.6	5,229,200.0
Makespan–InsertionMove	1,016.1	5,330,464.5
InsertionMove	1,020.6	5,299,200.0
Weightedmakespan–InsertionMove	1,010.5	5,201,021.8
BestInsertionMove	1,042.0	5,378,337.9
PALS	**1,001.5**	**5,186,352.2**

Table 6 Parameter setting for the grid job scheduling algorithm—α and β

α	β	Makespan		α	β	Makespan	
		B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)			B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
1	2	1,026.1	5,441,865.2	4	1	1,032.1	5,469,178.2
2	1	1,035.8	5,495,182.3	4	2	1,034.6	5,420,285.0
2	2	1,026.9	5,422,741.5	4	3	1,031.7	5,361,287.3
3	2	1,029.8	5,362,954.1	4	4	1,025.4	5,342,251.5
3	3	1,027.5	5,305,422.8	4	5	1,015.9	5,242,357.4
3	4	1,021.1	5,316,746.1	4	6	1,016.2	5,265,689.7
3	5	1,012.0	5,222,019.6	4	7	1,018.9	5,324,348.3
3	**6**	**1,001.5**	**5,186,352.2**	4	8	1,014.0	5,267,105.1

Table 7 Parameter setting for the crossover heuristic of grid job scheduling algorithm

γ	ϕ	Threshold	Makespan		γ	ϕ	Threshold	Makespan	
			B.u_s_lolo (2,048 × 32)	A.u_i_hihi (1,024 × 16)				B.u_s_lolo (2,048 × 32)	A.u_i_hihi (1,024 × 16)
3	5	0.0001	1,020.7	5,299,012.6	5	5	0.0001	1,024.0	5,247,216.0
3	5	0.0002	1,027.2	5,356,816.7	5	5	0.0002	1,026.3	5,253,171.1
3	5	0.0003	1,027.6	5,330,182.2	5	5	0.0003	1,014.6	5,223,572.8
3	10	0.0001	1,024.7	5,350,671.9	5	10	0.0001	1,026.6	5,288,196.0
3	10	0.0002	1,022.9	5,362,850.9	5	10	0.0002	1,021.9	5,222,340.7
3	10	0.0003	1,029.7	5,281,775.9	5	10	0.0003	1,017.5	5,203,137.7
3	15	0.0001	1,023.3	5,255,986.1	**5**	**15**	**0.0001**	**995.8**	**5,167,781.0**
3	15	0.0002	1,020.8	5,285,861.7	5	15	0.0002	1,016.1	5,216,907.6
3	15	0.0003	1,001.5	5,186,352.2	5	15	0.0003	1,020.4	5,216,143.2
3	20	0.0001	1,018.3	5,294,552.2	5	20	0.0001	1,018.2	5,233,202.7
3	20	0.0002	1,024.9	5,291,188.9	5	20	0.0002	1,021.6	5,230,685.2
3	20	0.0003	1,028.1	5,323,474.8	5	20	0.0003	1,017.9	5,267,414.8
3	25	0.0001	1,021.8	5,266,618.9	5	25	0.0001	1,021.6	5,256,397.7
3	25	0.0002	1,028.0	5,290,149.4	5	25	0.0002	1,012.2	5,245,060.9
3	25	0.0003	1,020.6	5,269,209.8	5	25	0.0003	1,012.6	5,297,869.4
4	5	0.0001	1,016.2	5,249,395.9	6	5	0.0001	1,017.0	5,234,716.3
4	5	0.0002	1,013.8	5,245,015.0	6	5	0.0002	1,021.0	5,255,787.8
4	5	0.0003	1,019.5	5,287,330.0	6	5	0.0003	1,020.9	5,224,064.8
4	10	0.0001	1,023.3	5,230,048.0	6	10	0.0001	1,023.8	5,283,304.3
4	10	0.0002	1,016.3	5,230,750.5	6	10	0.0002	1,011.3	5,205,000.1
4	10	0.0003	1,015.2	5,294,288.6	6	10	0.0003	1,009.7	5,199,400.5
4	15	0.0001	1,012.9	5,293,467.7	6	15	0.0001	1,016.8	5,243,789.8
4	15	0.0002	1,015.9	5,248,398.7	6	15	0.0002	1,023.8	5,292,174.1
4	15	0.0003	1,024.7	5,285,311.3	6	15	0.0003	1,017.1	5,288,434.8
4	20	0.0001	1,015.9	5,221,634.2	6	20	0.0001	1,014.0	5,240,307.8
4	20	0.0002	1,016.4	5,249,023.7	6	20	0.0002	1,018.4	5,217,704.9
4	20	0.0003	1,028.2	5,322,634.8	6	20	0.0003	1,021.7	5,235,787.7
4	25	0.0001	1,017.7	5,225,983.8	6	25	0.0001	1,024.5	5,241,498.1
4	25	0.0002	1,019.1	5,229,429.3	6	25	0.0002	1,026.8	5,200,877.4
4	25	0.0003	1,017.6	5,281,501.5	6	25	0.0003	1,017.2	5,211,054.3

Table 8 Importance of diversification phase

Organization of algorithm	Makespan	
	B.u_s_lolo (2,048 × 64)	A.u_i_hihi (1,024 × 32)
Only with exploration phase	1,042.6	5,601,953.9
Exploration + BVNS algorithm	1,024.6	5,261,099.2
Exploration + Crossover heuristic	1,044.5	5,487,141.6
Exploration + Local search(BVNS) + Crossover heuristic	1,019.2	5,306,646.6
Exploration + Shaking phase (BVNS) + Crossover heuristic	1,046.8	5,512,763.2
Exploration + Shaking phase(BVNS)	1,045.7	5,439,469.4
Exploration + Local search(BVNS)	1,037.5	5,416,292.4
Exploration + BVNS + Crossover heuristic	**995.8**	**5,167,781.0**

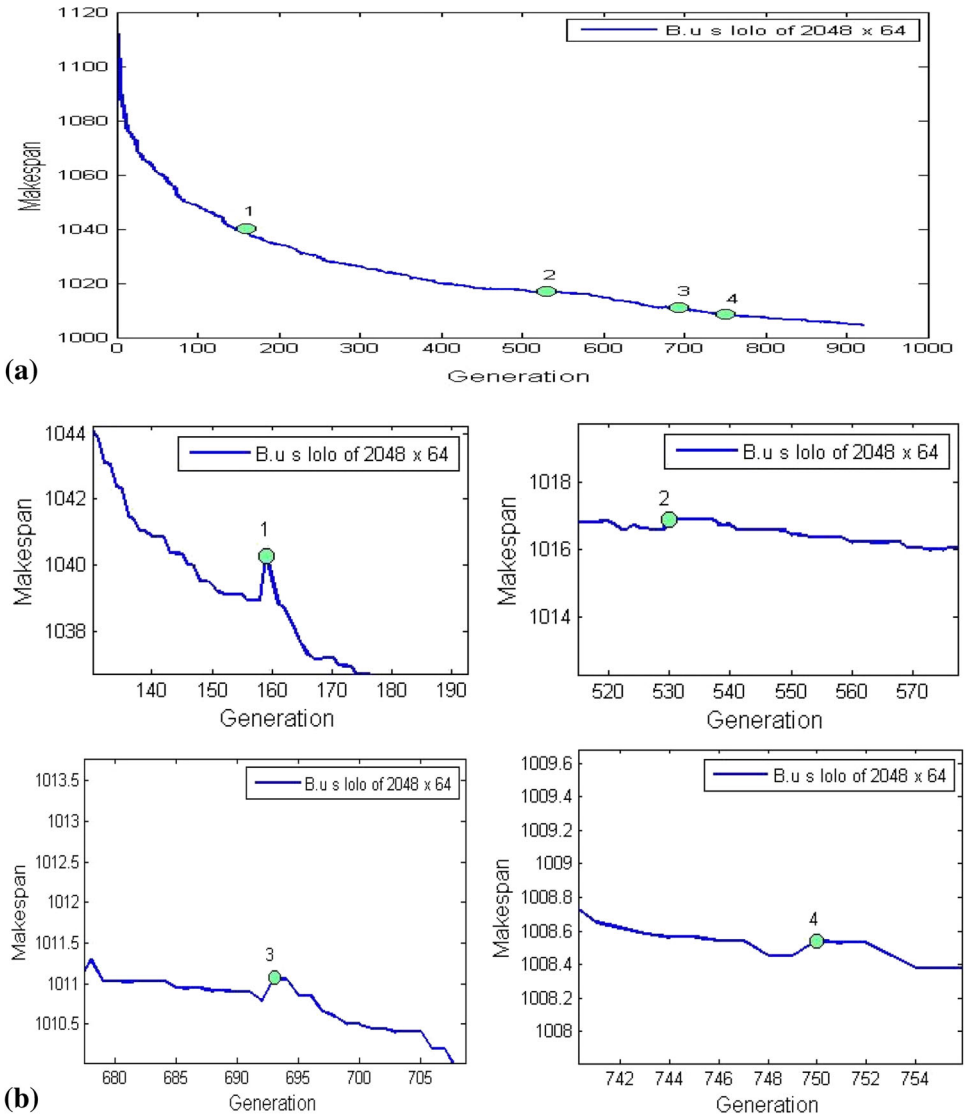

Fig. 5 **a** Evolution of makespan values observed for the TPVNS algorithm, **b** enlarged view of the occurrence of deteriorated solutions

and *threshold* are reported in Table 7. The grid job scheduling algorithm produces solution with minimum makespan while testing the algorithm by fixing the value of γ, ϕ and *threshold* at 5, 15, and 0.0001, respectively.

7.1.3 Contribution of Diversification Module to Grid Job Scheduling Algorithm

As diversification is considered as the part of the job scheduling algorithm, the relevancy of inclusion of that phase has to be justified. The result of the experiments carried out to examine the necessity of the diversification phase of job scheduling algorithm is given in Table 8. It is observed that the exclusion of diversification phase results in the generation of poor solution. Also the job scheduling algorithm had evolved through more number of generation when part of the diversification phase is excluded from the scheduling algorithm. It is revealed from Table 8 that the grid job scheduling algorithm requires two modules of diversification phase to find better mapping of jobs with grid nodes.

Figure 5 shows the evolution of makespan values observed for the VNS algorithm during the representative execution over the B.u_s_lolo test instance of the configuration 2,048 × 64. Few deteriorated solutions (marked as 1, 2, 3, and 4) generated by the diversification module are highlighted in Fig. 5a, and their enlarged views are shown in Fig. 5b. The deteriorated solution obtained at the 159th generation was improved immediately in the next generation. But the solution generated during the 530th generation took 10 more generations for getting the better solution. The deteriorated solution obtained in the 693rd generation diverts the scheduling algorithm to explore 3 different solutions, and then, the better mapping of grid jobs with grid nodes is found. The scheduling algorithm generated solution with makespan

1,008.54 at generation 746 and retained the same position for the next generation. Then, it finds the better mapping with makespan 1,008.45 and maintained same makespan value for the next generation also. At generation 750, the diversification module yielded the deteriorated solution with makespan 1,008.54. After that, the diversification module triggered the scheduling algorithm to search in the different direction to reach the makespan value 1,008.45 through the solution with makespan 1,008.53 and which in turn yields the schedule with makespan 1,008.38 at 754th generation. It is found that half of the decision making duration (t_{max}) is spent for the scheduling algorithm in the diversification module. Also it is noted that the diversification module is more beneficial for the scheduling algorithm to explore new better solutions until the makespan gap between lower bound and incumbent solution is greater than 10 %. If the gap is less than 10 %, the scheduling algorithm takes more than 10 generation to find a better solution from the deteriorated solution.

7.2 Results and Discussion

This section discusses the experimental results of applying the TPVNS algorithm to solve the grid job scheduling problem. In the computation experiments, 124 test instances were solved with the TPVNS algorithm. For each test instance, the TPVNS algorithm was run for 50 times and the best of these 50 runs are reported. 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 TPVNS algorithm. The overall best result produced by the TPVNS algorithm compared with all algorithms is represented in bold and italic. The relative gap value of the best makespan of TPVNS algorithm with respect to the correspondent lower bound is calculated using Eq. (10) (used in Sects. 7.2.1 and 7.2.2).

$$GAP\,(LB) = (result - LB)/LB \tag{10}$$

where *LB* denotes the lower bound reported in the literature [81] and *result* indicates the best makespan value obtained by the TPVNS algorithm for the correspondent test instance.

The improvement of an algorithm over another is computed using Eq. (11).

$$Improvement\,(\%) = \frac{\delta_1 - \delta_2}{\delta_2} \times 100\,\% \tag{11}$$

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

Table 9 Comparative makespan results: metaheuristics for 512 × 16 test instances from Braun et al. [5]

Instance	GA [5]	MA+TS [32]	cMA [20]	ACO+TS [23]	TS [27]	pCHC [8]		TPVNS			LB [81]	GAP (LB) (%)	Avg. GAP (LB) (%)
						Best	Avg.	Best	Avg.	σ (%)			
u_c_hihi.0	8,050,844.5	7,530,020.2	7,700,929.8	7,497,200.9	7,448,640.5	7,461,819.1	7,481,194.5	***7,439,471.8***	7,494,257.8	0.27	7,346,524.2	1.27	0.80
u_c_hilo.0	156,249.2	153,917.2	155,334.8	154,234.6	**153,263.3**	153,791.9	153,924.0	153,270.1	154,400.3	0.21	152,700.4	0.37	
u_c_lohi.0	258,756.8	245,288.9	251,360.2	244,097.3	241,672.7	241,524.0	243,446.3	***240,803.3***	244,043.2	0.19	238,138.1	1.12	
u_c_lolo.0	5,272.3	5,173.7	5,218.2	5,178.4	5,155.0	5,177.5	5,181.6	***5,154.8***	5,190.3	0.22	5,132.8	0.43	
u_i_hihi.0	3,104,762.5	3,058,474.9	3,186,664.7	2,947,754.1	2,957,854.1	2,952,493.2	2,956,905.7	***2,944,074.6***	2,955,764.7	0.25	2,909,326.6	1.19	0.84
u_i_hilo.0	75,816.1	75,108.5	75,856.6	73,776.2	73,692.9	73,639.8	73,847.1	***73,378.0***	73,927.0	0.23	73,057.9	0.44	
u_i_lohi.0	107,500.7	105,808.6	110,620.8	102,445.8	103,865.7	102,136.1	102,677.3	***102,057.5***	102,599.7	0.21	101,063.4	0.98	
u_i_lolo.0	2,614.4	2,596.6	2,624.2	2,553.5	2,552.1	2,549.8	2,557.2	***2,547.9***	2,560.1	0.26	2,529.0	0.75	
u_s_hihi.0	4,566,206.0	4,321,015.4	4,424,540.9	4,162,547.9	4,168,795.9	4,198,779.5	4,239,146.3	***4,145,941.7***	4,197,996.5	0.28	4,063,563.7	2.03	1.34
u_s_hilo.0	98,519.4	97,177.3	98,283.7	96,762.0	96,180.9	96,623.3	96,750.3	***95,872.3***	96,330.4	0.18	95,419.0	0.48	
u_s_lohi.0	130,616.5	127,633.0	130,014.5	123,922.0	123,407.4	123,251.5	123,989.4	***122,986***	123,954.3	0.25	120,452.3	2.10	
u_s_lolo.0	3,583.4	3,484.1	3,522.1	3,455.2	3,450.5	3,450.1	3,472.2	***3,440.5***	3,461.9	0.17	3,414.8	0.75	

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 10 Percentage improvement of TPVNS over other metaheuristics

Instance	GA [5]	MA+TS [32]	cMA [20]	ACO+TS [23]	TS [27]	pCHC [8]	Improvement based on consistency					
							GA [5]	MA+TS [32]	cMA [20]	ACO+TS [23]	TS [27]	pCHC [8]
u_c_hihi.0	7.59	1.20	3.40	0.77	0.12	0.30	4.67	0.95	2.53	0.8	0.12	0.34
u_c_hilo.0	1.91	0.42	1.33	0.63	−0.004	0.34						
u_c_lohi.0	6.94	1.83	4.20	1.35	0.36	0.30						
u_c_lolo.0	2.23	0.37	1.21	0.46	0.004	0.44						
u_i_hihi.0	5.18	3.74	7.61	0.12	0.47	0.29	4	2.87	5.38	0.32	0.7	0.2
u_i_hilo.0	3.22	2.30	3.27	0.54	0.43	0.36						
u_i_lohi.0	5.06	3.55	7.74	0.38	1.74	0.08						
u_i_lolo.0	2.54	1.88	2.91	0.22	0.16	0.07						
u_s_hihi.0	9.20	4.05	6.30	0.40	0.55	1.26	5.43	2.57	4.12	0.62	0.38	0.63
u_s_hilo.0	2.69	1.34	2.45	0.92	0.32	0.78						
u_s_lohi.0	5.84	3.64	5.42	0.76	0.34	0.22						
u_s_lolo.0	3.99	1.25	2.32	0.43	0.29	0.28						
Average	4.7	2.13	4.01	0.58	0.4	0.39						

Table 11 Makespan results for 1,024 × 32 test instances from Nesmachnow et al. [8]

Instance	pCHC [8]		TPVNS				LB [81]	GAP (LB) (%)	Avg. GAP (LB) (%)
	Best	Avg.	Best	Avg.	σ(%)	Impr. (%)			
A.u_c_hihi	20,327,924.0	20,510,300.9	***20,194,902.0***	20,284,191.0	0.27	0.65	19,449,230.0	3.83	4.44
A.u_c_hilo	2,048,582.7	2,058,352.2	***2,046,648.0***	2,050,942.2	0.17	0.09	1,951,345.0	4.88	
A.u_c_lohi	**1,956.7**	2,000.0	1,962	1,970.2	0.32	−0.27	1,866.4	5.12	
A.u_c_lolo	207.5	217.8	***206.7***	213.4	0.19	0.39	198.9	3.92	
A.u_i_hihi	5,169,960.5	5,244,046.9	***5,167,781.0***	5,221,702.0	0.23	0.04	5,012,207	3.10	3.44
A.u_i_hilo	490,280.3	492,699.4	***489,525.2***	493,800.1	0.20	0.15	474,404.6	3.19	
A.u_i_lohi	**518.2**	523.6	522.4	530.1	0.41	−0.80	503.4	3.77	
A.u_i_lolo	**50.6**	51.7	50.8	51.9	0.31	−0.41	49.0	3.69	
A.u_s_hihi	12,243,560.0	12,439,843.1	***12,155,750***	12,306,122.0	0.12	0.72	11,553,632.0	5.21	4.91
A.u_s_hilo	1,187,506.4	1,214,303.0	***1,175,338***	11,885,443.2	0.16	1.02	1,126,556.0	4.33	
A.u_s_lohi	1,186.8	1,199.2	***1,184.8***	1,194.6	0.25	0.17	1,122.2	5.57	
A.u_s_lolo	122.4	126.5	***122.0***	123.1	0.19	0.33	116.7	4.54	
B.u_c_hihi	**6,169,823.0**	6,200,118.0	6,189,681	6,200,401.5	0.20	−0.32	5,980,872	3.49	3.22
B.u_c_hilo	61,114.7	61,390.1	***60,807.5***	61,599.2	0.29	0.50	58,942.5	3.16	
B.u_c_lohi	215,149.2	218,124.8	***214,387.1***	216,481.5	0.31	0.35	207,892.8	3.12	
B.u_c_lolo	2,164.3	2,208.4	***2,142.1***	2,158.3	0.19	1.03	2,078.0	3.08	
B.u_i_hihi	1,630,288.6	1,670,112.7	***1,626,086***	1,628,729.6	0.29	0.26	1,567,179	3.76	3.33
B.u_i_hilo	15,121.5	15,464.1	***15,003.1***	15,715.8	0.21	0.78	14,582.3	2.89	
B.u_i_lohi	49,569.9	50,128.2	***49,264.1***	49,981.2	0.24	0.62	47,606.9	3.48	
B.u_i_lolo	496.1	507.4	***492.7***	501.4	0.18	0.69	477.4	3.20	
B.u_s_hihi	3,393,010.2	3,430,218.1	***3,344,875***	3,392,157.3	0.17	1.42	3,178,482	5.23	4.81
B.u_s_hilo	35,988.4	36,515.6	***35,352.2***	36,911.2	0.21	1.77	33,948.7	4.13	
B.u_s_lohi	115,179.2	118,070.3	***114,653.3***	117,017.1	0.25	0.46	108,330.1	5.84	
B.u_s_lolo	1,191.7	1,230.3	***1,173.5***	1,196.4	0.19	1.53	1,128.1	4.02	

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 12 Makespan results for 2,048 × 64 test instances from Nesmachnow et al. [8]

Instance	pCHC [8]		TPVNS				LB [81]	GAP (LB) (%)	Avg. GAP (LB) (%)
	Best	Avg.	Best	Avg.	σ(%)	Impr. (%)			
A.u_c_hihi	18,110,479.1	18,218,285.6	***17,795,863.0***	17,801,492.2	0.21	1.74	17,141,977.4	3.81	3.76
A.u_c_hilo	1,748,509.2	1,760,141.2	***1,727,248.0***	1,736,971.3	0.15	1.22	1,664,592.8	3.76	
A.u_c_lohi	1,798.4	1,804.9	***1,761.0***	1,770.6	0.29	2.08	1,695.3	3.88	
A.u_c_lolo	177.6	178.1	***174.3***	175.5	0.22	1.86	168.3	3.57	
A.u_i_hihi	2,506,258.5	2,546,459.7	***2,478,011.0***	2,500,937.2	0.33	1.13	2,366,682.1	4.70	5.11
A.u_i_hilo	**272,741.3**	273,876.3	274,378.4	276,000.1	0.23	−0.60	260,904.5	5.16	
A.u_i_lohi	266.3	267.5	***265.9***	266.2	0.35	0.15	255.2	4.19	
A.u_i_lolo	**26.4**	26.5	26.7	26.9	0.19	−1.13	25.1	6.37	
A.u_s_hihi	9,756,499.7	9,821,934.5	***9,524,603.0***	9,601,364	0.17	2.38	9,050,260.8	5.24	5.06
A.u_s_hilo	924,094.9	937,998.8	***894,695.3***	909,381.6	0.22	3.18	851,399.9	5.09	
A.u_s_lohi	947.1	952.3	***931.6***	936.1	0.13	1.64	888.9	4.80	
A.u_s_lolo	99.6	100.4	***97***	98.9	0.15	2.61	92.3	5.09	
B.u_c_hihi	5,290,128.2	5,300,316.1	***5,209,573.0***	5,219,961.3	0.19	1.52	4,975,778.8	4.70	4.24
B.u_c_hilo	55,316.2	55,343.1	***53,960.3***	54,001.5	0.33	2.45	52,240.6	3.30	
B.u_c_lohi	177,063.4	177,612.4	***175,429.4***	176,981.2	0.22	0.92	167,381.1	4.80	
B.u_c_lolo	1,814.7	1,818.3	***1,786.3***	1,791.0	0.34	1.56	1,715	4.16	
B.u_i_hihi	770,110.6	774,993.0	***765,966.9***	769,121.1	0.24	0.54	735,101.5	4.20	5.08
B.u_i_hilo	7,906.5	7,932.9	***7,896.9***	7,910.1	0.18	0.12	7,536.3	4.78	
B.u_i_lohi	**26,941.2**	27,207.3	27,118.9	27,900.4	0.26	−0.66	25,681.2	5.60	
B.u_i_lolo	**262.4**	264.7	264.9	265.8	0.30	−0.95	250.5	5.75	
B.u_s_hihi	2,910,507.6	2,923,857.1	***2,865,250.0***	2,876,310.0	0.25	1.55	2,710,024.0	5.73	5.20
B.u_s_hilo	29,442.2	29,518.6	***28,520.4***	28,731.2	0.12	3.13	27,268.0	4.59	
B.u_s_lohi	98,607.0	98,758.3	***94,777.9***	95,101.4	0.10	3.88	90,727.3	4.46	
B.u_s_lolo	1,014.3	1,019.7	***995.8***	1,003.2	0.21	1.82	939.0	6.05	

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 13 Makespan results for 4,096 × 128 test instances from Nesmachnow et al. [8]

Instance	pCHC [8]		TPVNS				LB [81]	GAP (LB) (%)	Avg. GAP (LB) (%)
	Best	Avg.	Best	Avg.	σ (%)	Impr. (%)			
A.u_c_hihi	15,722,681.0	15,760,840.0	***15,418,041.0***	15,701,562.7	0.27	1.94	14,829,360.6	3.97	4.53
A.u_c_hilo	1,562,810.9	1,565,580.1	***1,550,224.0***	1,559,100.0	0.16	0.81	1,478,358.1	4.86	
A.u_c_lohi	1,540.9	1,545.1	***1,515.9***	1,520.3	0.17	1.62	1,452.5	4.36	
A.u_c_lolo	155.7	156.2	***154.7***	160.1	0.12	0.66	147.4	4.94	
A.u_i_hihi	1,309,493.5	1,331,529.0	***1,307,049.0***	1,319,121.4	0.35	0.19	1,231,099	6.17	8.21
A.u_i_hilo	137,158.4	139,250.8	***137,022.0***	138,711.3	0.11	0.10	128,539.5	6.60	
A.u_i_lohi	**136.1**	137.7	139.3	140.7	0.22	−2.35	127.6	9.17	
A.u_i_lolo	**13.7**	13.7	14.3	15.0	0.25	−4.45	12.9	10.93	
A.u_s_hihi	8,089,853.5	8,121,957.0	***7,943,696.0***	7,991,425.0	0.27	1.81	7,553,763.4	5.16	6.54
A.u_s_hilo	828,912.4	834,878.5	***811,887.1***	819,912.3	0.15	2.05	768,703.1	5.62	
A.u_s_lohi	**807.6**	811.9	809.3	810.1	0.29	−0.21	748.5	8.12	
A.u_s_lolo	84.2	84.5	***84.1***	86.2	0.14	0.12	78.4	7.27	
B.u_c_hihi	4,767,774.5	4,789,005.9	***4,709,910.0***	4,720,001.1	0.19	1.21	4,514,305.9	4.33	3.97
B.u_c_hilo	46,350.1	46,470.8	***45,782.2***	46,001.2	0.22	1.23	44,027	3.99	
B.u_c_lohi	158,780.8	159,312.0	***156,567.0***	157,813.4	0.18	1.39	150,530	4.01	

Table 13 continued

Instance	pCHC [8]		TPVNS				LB [81]	GAP (LB) (%)	Avg. GAP (LB) (%)
	Best	Avg.	Best	Avg.	σ (%)	Impr. (%)			
B.u_c_lolo	1,556.8	1,562.2	***1,526.3***	1,529.1	0.10	1.96	1,474	3.55	
B.u_i_hihi	**402,182.1**	405,768.5	405,708.8	407,911.5	0.32	−0.88	374,988.9	8.19	9.53
B.u_i_hilo	**4,224.8**	4,252.2	4,418.8	4,490.8	0.24	−4.59	3,942.5	12.08	
B.u_i_lohi	13,847.8	13,905.8	***13,760.9***	14,250.2	0.21	0.63	12,825.3	7.29	
B.u_i_lolo	**137.4**	138.9	142.4	143.6	0.24	−3.63	128.8	10.56	
B.u_s_hihi	2,508,467.3	2,524,194.9	***2,362,473.0***	2,400,121.0	0.17	5.82	2,353,555.3	0.38	5.88
B.u_s_hilo	**25,244.1**	25,346.6	25,353.3	25,491.2	0.13	−0.43	23,417.3	8.27	
B.u_s_lohi	81,118.5	81,559.4	***80,644.5***	81,000.1	0.27	0.58	75,488.9	6.83	
B.u_s_lolo	**825.7**	830.9	834.0	838.4	0.32	−1.00	771.8	8.06	

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 14 Percentage improvement of TPVNS over pCHC

Model	Type	Improvement (%)			Overall improvement (%)		
		1,024 × 32	2,048 × 64	4,096 × 128	1,024 × 32	2,048 × 64	4,096 × 128
Ali et al.	Consistent	0.22	1.72	1.26	0.17	1.35	0.19
	Inconsistent	−0.26	−0.11	−1.62			
	Semiconsistent	0.56	2.45	0.94			
Braun et al.	Consistent	0.39	1.62	1.45	0.76	1.33	0.19
	Inconsistent	0.59	−0.24	−2.12			
	Semiconsistent	1.29	2.60	1.24			

Table 15 Comparative makespan results of test instances from Liu et al. [24]

Instance	GA [24]	SA [24]	fuzzyPSO [24]	DE [25]	TPVNS		
	Avg.	Avg.	Avg.	Avg.	Best	Avg.	σ (%)
(3,13)	47.1167	46.6000	46.2667	**46.05**	46.0000	46.2500	0.11
(8,60)	42.9270	55.4594	41.9489	42.48	41.7227	***41.7412***	0.12
(5,100)	85.7431	90.7338	**84.0544**	86.36	85.4345	85.4357	0.10
(10,50)	38.0428	41.7889	37.6668	38.39	35.1586	***35.2478***	0.13

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

7.2.1 Results for the Problem Instances of Braun et al. [5]

The results of the computational experiments to demonstrate the performance of the proposed TPVNS algorithm compared with the former studies, for the set of instances of 512 × 16 configuration proposed by Braun et al., are presented in this section. Table 9 reports the best makespan values of GA, MA+TS, cMA, ACO+TS, TS, pCHC, and TPVNS for 12 test instances. The best, average, and standard deviation on the makespan results achieved during the experimentation of TPVNS algorithm are given in column 9, 10, and 11 of Table 9, respectively. The average makespan result of pCHC is given in column 8 of Table 9. From Table 9, it is observed that the results for 11 out of 12 instances are improved.

In columns 12, 13, and 14 of Table 9, the lower bound values reported in the literature [81], the percentage gap value, and the average deviation from the lower bounds for consistent, inconsistent and semiconsistent, respectively, are reported. The relative gap values are below 1 % for 7 test instances, around 1.2 % for 3 test instances, and around 2 % for 2 test instances. The average gap value is around 0.8 % for consistent and inconsistent test scenarios and around 1.4 % for semiconsistent test scenarios.

Table 10 summarizes the percentage improvement of TPVNS algorithm over other metaheuristics. The makespan improvement factors are greater than 4 % with respect to GA and cMA, around 2 % with respect to MA+TS, around 0.4 % with respect to TS and pCHC, and around 0.5 % with respect to ACO+TS. Lower improvement factors are obtained with respect to ACO+TS, TS, and pCHC methods. Regarding the consistency classification, TPVNS obtained slight improvements over pCHC for inconsistent scenarios,

but the improvements in consistent and semiconsistent scenarios are slightly better than the inconsistent scenarios. The analysis of Tables 9 and 10 shows that TPVNS is able to compute better makespan values than other metaheuristics, outperforming the ACO+TS [23], the TS [27], and the pCHC [8] for solving the instances by Braun et al.

7.2.2 Results for Instances from Nesmachnow et al. [8]

This section presents the results for the large heterogeneous scheduling computing problem instances by Nesmachnow et al. for the configuration $1{,}024 \times 32$, $2{,}048 \times 64$, and $4{,}096 \times 128$. For each configuration, the results for the twelve instances following the heterogeneity model from Ali et al. and the twelve ones following the heterogeneity model from Braun et al. are presented.

Tables 11, 12, and 13 present the best and average makespan values obtained with pCHC, the best, average, and standard deviation results achieved using TPVNS algorithm in 50 independent executions, and the percentage improvement factors over the pCHC algorithm result. The lower bound value reported in the literature [81], the percentage gap value, and the average gap value for the consistent, inconsistent, and semiconsistent scenarios are also mentioned in the columns 8, 9, and 10 of Tables 11, 12, and 13, respectively.

From Table 11, it is inferred that the percentage improvement factor of TPVNS over pCHC for the $1{,}024 \times 32$ configuration is slightly better for test instances of Braun et al. model than the Ali et al. model. The average gap value is around 5 % for inconsistent scenario and around 3 % for consistent scenario of both models and around 3 and 4 % for consistent scenario of Braun et al. and Ali et al. models, respectively.

It is observed from Table 12 that the improvement factor of TPVNS over pCHC is around 1 % for 9 test instances of $2{,}048 \times 64$ configuration. But the average gap value varies from around 3.5–5 % for various consistency configurations.

The TPVNS algorithm gave the best mapping of grid jobs with grid nodes for the test instance B.u_s_hihi of $4{,}096 \times 128$ configuration. The improvement factor of TPVNS over pCHC for that test instance is around 6 %, and the average gap value of TPVNS with respect to the concern lower bound is also very less (0.38 %) when compared to other test instances proposed by Nesmachnow et al. The average gap value varies from around 4–9.5 % for various consistency configurations of $4{,}096 \times 128$ dimension, which is reported in Table 13.

The percentage gap value for semiconsistent configuration is usually greater than other consistency configuration except for $4{,}096 \times 128$ dimension. The gap value for inconsistent is greater than semiconsistent for $4{,}096 \times 128$ and is greater than consistent for $2{,}048 \times 64$ configuration. It is observed from Tables 11, 12, and 13 that the percentage gap value for inconsistent and semiconsistent scenarios increases with the increase in the size of the problem. The

Table 16 Makespan results for 100×10 additional test instances

Instance	Min–Min	S A				GRASP			TPVNS						
		With Min–Min seed	With random seed						With Min–Min seed	With random seed			Impr. over (%)		
			Best	Average	σ (%)	Best	Average	σ (%)		Best	Average	σ (%)	S A	GRASP	Min–Min
c_hihi	3,537,079.1	3,077,144.9	4,061,863.5	4,370,539.1	0.23	4,068,660.1	4,141,418.3	0.31	2,888,742.6	***2,961,418.1***	3,017,246.4	0.21	27.09	27.21	16.28
c_hilo	49,807.0	44,816.3	56,308.9	59,540.2	0.17	54,000.3	55,984.7	0.25	42,873.2	***42,211.9***	42,700.2	0.24	25.04	21.83	15.25
c_lohi	113,580.2	102,011.6	124,861.5	129,990.1	0.19	136,243.5	138,047.9	0.29	93,670.8	***95,935.5***	97,139.1	0.20	23.17	29.59	15.54
c_lolo	1,469.1	1,350.7	1,706.4	1,795.9	0.21	1,619.2	1,666.5	0.24	1,283.1	***1,298.7***	1,351.9	0.18	23.93	19.83	11.64
i_hihi	2,264,784.0	1,814,449.2	1,831,121.9	1,891,663.2	0.12	3,395,177.8	3,615,236.7	0.19	1,738,978.1	***1,690,693.6***	1,746,829.2	0.19	7.67	50.20	25.35
i_hilo	30,899.2	23,558.6	26,644.0	29,041.2	0.15	44,251.1	48,233.0	0.20	23,662.0	***23,275.2***	23,796.0	0.22	12.64	47.40	24.67
i_lohi	71,098.4	55,305.6	57,187.9	59,016.8	0.22	96,102.0	99,600.8	0.28	51,871.9	***52,538.5***	54,274.3	0.18	8.13	45.33	26.10
i_lolo	929.0	769.2	830.5	890.9	0.11	1,260.9	1,330.3	0.24	772.4	***762.0***	810.2	0.16	8.25	39.52	17.98
s_hihi	2,657,872.1	2,173,774.4	2,364,038.4	2,531,807.6	0.28	2,385,450.3	2,569,662.7	0.19	2,072,588.8	***2,103,842.3***	2,215,165.6	0.22	11.00	11.80	20.84
s_hilo	33,399.4	29,034.5	36,239.1	38,895.3	0.18	40,045.9	41,096.9	0.27	27,963.3	***28,385.7***	31,176.7	0.19	21.67	29.12	15.01
s_lohi	83,046.6	68,867.9	78,263.8	80,409.9	0.22	117,418.4	124,032.0	0.18	63,016.4	***64,279.3***	69,410.9	0.18	17.87	45.26	22.59
s_lolo	1,191.3	967.4	1,063.8	1,167.9	0.15	970.1	1,192	0.16	925.7	***924.6***	980.0	0.20	13.14	4.74	22.42

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 17 Makespan results for 300 × 10 additional test instances

Instance	Min–Min	S A				GRASP			TPVNS						
		With Min–Min seed	With random seed						With Min–Min seed	With random seed					
			Best	Average	σ (%)	Best	Average	σ (%)		Best	Average	σ (%)	Impr. over (%)		
													S A	GRASP	Min–Min
c_hihi	9,522,427.0	8,843,712.4	12,182,189.9	12,378,682.3	0.25	12,266,073.3	12,302,114.3	0.21	8,291,574.1	***8,458,026.0***	8,612,523.1	0.13	30.57	31.05	11.18
c_hilo	123,209.2	119,780.6	162,703.7	168,560.6	0.19	153,130.1	154,165.2	0.18	117,215.3	***118,640.4***	120,563.6	0.11	27.08	22.52	3.71
c_lohi	341,632.3	323,877.4	441,582.6	451,793.2	0.23	442,047.0	449,881.7	0.20	300,302.4	***304,260.1***	312,995.2	0.18	31.09	31.17	10.93
c_lolo	4,174.2	4,007.9	5,467.9	5,616.4	0.20	5,179.6	5,406.1	0.19	3,933.0	***3,951.5***	4,201.1	0.20	27.73	23.71	5.33
i_hihi	4,882,378.4	4,295,119.9	5,285,328.0	5,344,560.6	0.18	10,798,251.2	10,886,953.6	0.16	4,148,445.6	***4,175,679.3***	4,292,435.7	0.19	20.99	61.33	14.47
i_hilo	72,525.1	66,831.6	80,335.0	84,716.5	0.16	122,562.4	124,690.0	0.19	65,178.2	***65,295.4***	66,975.0	0.23	18.72	46.72	9.97
i_lohi	199,553.3	166,429.4	198,181.1	201,333.2	0.19	383,453.5	387,571.4	0.14	161,502.4	***161,245.8***	163,368.6	0.21	18.63	57.94	19.19
i_lolo	2,436.1	2,288.4	2,886.3	3,155.8	0.21	4,072.3	4,368.5	0.23	2,211.5	***2,186.1***	2,421.7	0.19	24.26	46.32	10.26
s_hihi	6,248,126.0	5,632,547.9	7,735,043.5	7,976,403.2	0.24	9,051,046.7	9,432,546.3	0.25	5,472,187.8	***5,447,286.0***	5,476,580.3	0.18	29.58	39.82	12.82
s_hilo	87,960.2	80,860.8	105,918.7	114,272.3	0.20	120,360.0	122,689.1	0.23	78,525.1	***79,199.6***	80,768.1	0.22	25.23	34.20	9.96
s_lohi	232,175.1	81,677.6	271,017.4	287,267.8	0.19	274,230.5	280,523.7	0.21	200,686.0	***197,800.2***	217,828.8	0.24	27.02	27.87	14.8
s_lolo	2,919.2	2,816.8	3,765.5	3,921.4	0.17	4,344.0	4,662.2	0.25	2,763.5	***2,758.4***	3,368.4	0.23	26.75	36.50	5.50

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

Table 18 Makespan results for 512 × 16 additional test instances

Instance	Min–Min	S A				GRASP			TPVNS						
		With Min–Min seed	With random seed						With Min–Min seed	With random seed					
			Best	Average	σ (%)	Best	Average	σ (%)		Best	Average	σ (%)	Impr. over (%)		
													S A	GRASP	Min–Min
c_hihi	8,323,054.0	7,937,713.1	12,723,933.8	12,756,472.5	0.21	13,237,637.1	13,969,099.2	0.15	7,643,333.1	***8,286,311.3***	8,337,617.1	0.19	34.88	37.40	0.44
c_hilo	139,009.1	133,146.6	185,618.8	189,017.6	0.30	146,768.0	154,948.6	0.18	131,191.8	***131,665.8***	134,557.4	0.18	29.07	10.29	5.28
c_lohi	272,900.4	264,104.6	426,733.5	451,557.9	0.25	481,261.3	498,347.4	0.25	261,121.5	***272,403.6***	282,640.0	0.16	36.17	43.40	0.18
c_lolo	4,310.2	4,141.7	5,690.3	6,033.5	0.19	5,731.6	6,103.7	0.22	41,120.4	***4,157.2***	4,397.6	0.20	26.94	27.47	3.55
i_hihi	3,292,135.1	3,079,656.6	7,665,632.6	7,958,603.8	0.21	10,430,504.4	10,665,410.6	0.21	2,966,035.3	***2,989,563.9***	3,065,324.4	0.16	61.00	71.34	9.19
i_hilo	65,978.4	62,216.3	110,812.2	114,938.8	0.23	164,241.2	169,348.7	0.24	61,392.5	***62,346.4***	63,364.3	0.17	43.74	62.04	5.50
i_lohi	112,106.3	104,617.1	265,837.7	276,321.5	0.21	365,416.8	390,862.8	0.21	100,386.9	***102,823.2***	104,378.2	0.19	61.32	71.86	8.28
i_lolo	2,130.0	2,015.3	3,847.5	4,306.9	0.20	5,407.3	5,957.5	0.19	1,947.2	***1,954.3***	2,157.8	0.21	49.21	63.86	8.25
s_hihi	4,670,099.1	4,297,614.8	10,509,710.9	10,867,948.0	0.19	10,430,504.4	10,722,278.3	0.15	4,204,172.0	***4,386,105.1***	4,400,068.1	0.23	58.27	57.95	6.08
s_hilo	86,913.3	82,504.6	140,744.6	148,224.3	0.21	151,715.0	179,619.4	0.18	80,434.4	***82,226.0***	83,697.3	0.21	41.58	45.80	5.39
s_lohi	158,003.2	147,631.9	368,494.8	388,772.7	0.18	472,916.7	491,135.7	0.17	143,276.8	***151,749.7***	152,967.2	0.23	58.82	67.91	3.96
s_lolo	2,639.0	2,516.9	4,715.5	4,960.8	0.17	5,442.6	6,061.2	0.16	2,477.3	***2,574.6***	2,824.6	0.20	45.40	52.70	2.44

Bold italic values indicate the overall best solution generated by the TPVNS algorithm

TPVNS algorithm finds better solution for the consistent test scenario with moderate gap value compared with inconsistent and semiconsistent test scenario and also does not have abrupt change in the gap value even though the problem size increases. It is inferred from Tables 11, 12, and 13 that the TPVNS algorithm produces a better average makespan value than the pCHC algorithm for 69.5 % of the test instances of Nesmachnow et al.

Table 14 gives the information about the percentage improvement factor of TPVNS over pCHC for different configurations of Braun et al. and Ali et al. models. The overall improvement factor is greater (around 1.3 %) for both models of 2,048 × 64 configuration and less (0.17 %) for Ali et al. model of 1,024 × 32 configuration. It is observed from Table 14 that the TPVNS algorithm has better percentage improvement factor over pCHC for consistent and semiconsistent scenario compared with inconsistent scenario. The TPVNS algorithm outperforms pCHC algorithm in producing better results for 56 test instances out of 72 instances of Nesmachnow et al.

7.2.3 Results for Instances from Liu et al. [24]

PSO [24] and DE [25] algorithms were experimented with the test instance of configurations (3, 13), (8, 60), (5,100), and (10, 50) proposed by Liu et al., and the average makespan for 10 different trials for each instance was reported. There is no information provided regarding the best makespan obtained during the experimentation of proposed algorithms for different 10 trials in both papers. Thus, the average makespan obtained for 50 different trials during the experimentation of TPVNS was considered for comparison with the results of DE [25] and with the work published by Liu et al. [24].

Table 15 reports the average makespan values of GA, SA, PSO, and DE, and the best, average, and standard deviation of makespan values obtained during the experimentation of TPVNS for each configuration. TPVNS had better average makespan values for the configurations (8,60) and (10,50) when compared to other metaheuristics. From Table 15, it is observed that the results for 2 out of 4 instances are improved.

7.2.4 Results for the Additional Problem Instances

Tables 16, 17, and 18 show the TPVNS result for 100 × 10, 300 × 10, and 512 × 16 configurations. The TPVNS 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 A.10) in the local search phase, in which

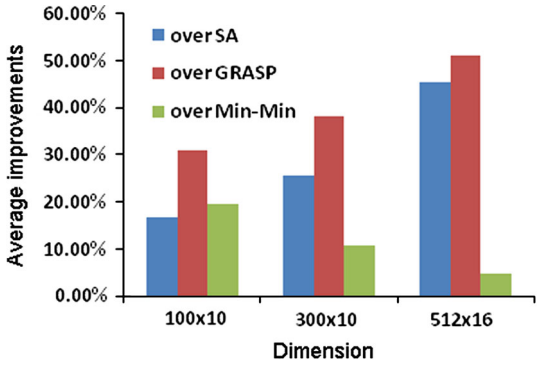

Fig. 6 Average improvements of TPVNS over other heuristics

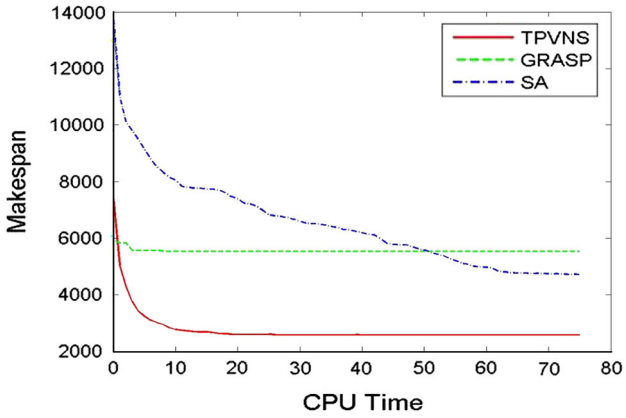

Fig. 7 Convergence of TPVNS, GRASP, and SA

PALS_maxiter and threshold parameter are set to 50 and 0.2, respectively.

The simulated annealing and TPVNS algorithm were run with Min–Min seed and random seed for all test instances. The best makespan values of SA and TPVNS with Min–Min seed are noted in the column 3 and 10 of Tables 16, 17, and 18. The best, average, and standard deviation of SA and TPVNS with random seed and GRASP were also reported. The TPVNS algorithm produces a good quality schedule for all newly generated test instances. The percentage improvement of VNS with random seed over SA with random seed, GRASP, and Min–Min are reported in column 14, 15, and 16 of Tables 16, 17, and 18, respectively.

Figure 6 shows the average improvement of TPVNS over the heuristic algorithms for the additional test instances of different configurations. It is revealed from Fig. 6 that the percentage improvement of TPVNS is gradually increased when the problem dimension grows for SA and GRASP. Even though TPVNS has better improvement over Min–Min algorithm, the percentage of improvement is gradually decreased for increasing problem dimension.

Figure 7 illustrates the convergence of algorithms for s_lolo test instance of 512 × 16 configuration. The convergence speed of TPVNS is better than SA.

Table 19 Comparison of TPVNS over other metaheuristics for all test instances

	Braun et al. [5]							Nesmachnow et al. [8]	Liu et al. [24]				Additional test Instances			
	GA [5]	MA+TS [32]	cMA [20]	ACO+TS [23]	TS [27]	pCHC [8]	pCHC [8]	GA [24]	SA [24]	Fuzzy PSO [24]	DE [25]	SA	GRASP	Min–Min
Number of improved solutions	12	12	12	12	11	12	56	4	4	3	3	36	36	36
Number of inferior solutions	–	–	–	–	1	–	16	–	–	1	1	–	–	–
Total solutions	12						72	4				36		

7.2.5 Summary

The pairwise comparison of the results of TPVNS algorithm with other reported algorithms is given in Table 19, which exhibits the number of improved and unimproved solutions of the TPVNS algorithm with respect to other algorithms. TPVNS improves the makespan of 105 test instances out of 124 test instances considered for the experimentation. In order to have a fair comparison with the previously published algorithms, a clear comparison concerns not only the best solution value achieved by the TPVNS, but also the average one. It is found that TPVNS algorithm achieves a better average makespan value than the best value of the reported algorithms of this article for 70 % of test instances considered for the experimentation.

8 Conclusion

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 a novel two-phase heuristic based on VNS algorithm for grid job scheduling problem to minimize the makespan. Extensive computational experiments have been devised to select a proper neighborhood order as well as to decide the parameters of the proposed algorithm. The performance of TPVNS 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 TPVNS are better for most of the instances. The computational results demonstrate the value of the proposed TPVNS 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.

Appendix A

Algorithm A.1. $f_1(x, ETC[\,][\,])$ /*fitness evaluation */

Input: $x, ETC[\,][\,]$

Output: $localmakespan[\,], fitnes$

1. $[n\ , m] \leftarrow size(ETC\ [\,][\,])$ /* n- number of jobs; m – number of grid nodes */
2. $localmakespan[\,] \leftarrow 0$ /* Reset the local makespan of all grid nodes */
3. ***for*** $i = 1$ to n ***do***
4. $localmakespan(x[i]) \leftarrow localmakespan(x[i]) + ETC(i, x[i])$
5. ***endfor***
6. $fitnes \leftarrow maximum(localmakespan[\,])$ /* Return the makespan */

Algorithm A.2. $f_2(x, ETC[\,][\,])$ /*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 A.3. Min-Min algorithm

Input: $ETC[\,][\,]$
Output: x

1 $[n, m] \leftarrow size(ETC[\,][\,])$
2 $readytime[\,] \leftarrow 0$
3 ***for*** $i = 1$ to n ***do***
4 ***for*** $j = 1$ to m ***do***
5 completion_time(i,j)= $ETC(i, j)$ + readytime[j];
6 ***endfor***
7 $Best \leftarrow$ minimum(completion_time(i,j))
8 readytime(Best) ← completion_ time(i,Best) /* Assign the node which has minimum completion time of i^{th} job to Best */
9 x[i] ← *Best*
10 ***endfor***

Algorithm A.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 A.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_l from the job list assigned for *Heavy*
4 x' ←Assign *Light* to J_l

Algorithm A.6. InsertionMove

Input: x

Output: x'

1 Select a random job J_l in J
2 Select a random resource G_l in G
3 x' ←Assign G_l to J_l

Algorithm A.7. Weightedmakespan –InsertionMove

Input: x, $ETC[\,][\,]$

Output: x'

1 Evaluate the fitness of x
2 Select two random nodes Lr and Hr ,
where $Localmakespan(Lr) \leq 0.25 \times fitnes$, and
$Localmakespan(Hr) \geq 0.75 \times fitnes$
3 Select a random job J_l from the job list assigned for Hr
4 x' ←Assign Lr to J_l

Algorithm A.8. 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_l in G, where G_l has minimum execution time for J_l
4 x' ←Assign G_l to J_l

Algorithm A.9. Problem Aware Local Search

Input: x, $ETC[\,][\,]$

Output: x'

1 Evaluate the fitness of x
2 $Best \leftarrow \infty$
3 $JJ[\,] \leftarrow$Job list of *Heavy*
4 Select a random node G_1, where $G_1 \neq Heavy$
5 $JJJ[\,] \leftarrow$Job list of G_l
6 $ln \leftarrow length(JJ[\,])$ /* Assign the total number of jobs of node *Heavy* to *ln* */
7 $lnr \leftarrow length(JJJ[\,])$ /* Assign the total number of jobs of node G_l to *lnr* */
8 $startheavy \leftarrow randi(1, ln - 1)$ /* Select random integer between 1 and *ln*-1 */
9 $endheavy \leftarrow randi(startheavy, ln)$
10 $startres \leftarrow randi(1, lnr - 1)$
11 $endres \leftarrow randi(startres, lnr)$
12 ***for*** $i = startheavy$ to $endheavy$ ***do***
13 ***for*** $j = startres$ to $endres$
14 $x'' \leftarrow$Swap the resources assigned for $JJ[i]$ and $JJJ[i]$
15 $[localmakespan\ , temp] \leftarrow$ Evaluate the fitness of x''
16 ***if*** ($temp < Best$)
17 $x' \leftarrow x''$
18 $Best \leftarrow temp$
19 ***endif***
20 ***endfor***
21 ***endfor***

Algorithm A.10. Problem Aware Local Search Heuristic

Input: $x, ETC[\,][\,], PALS\_maxiter$
Output: x

```
for i = 1 to PALS_maxiter do
  [localmakespan[ ], fitnes] ← f2(x, ETC[ ][ ])        /* Algorithm A.2 */
  Best ← ∞
  JJ[ ] ←Job list of Heavy
  Select a random node G1, where G1 ≠ Heavy
  JJJ[ ] ←Job list of Gl
  ln ← length(JJ[ ])
  lnr ← length(JJJ[ ])
  startheavy ← randi(1, ln − 1)
  endheavy ← randi(startheavy, ln)
  startres ← randi(1, lnr − 1)
  endres ← randi(startres, lnr)
  for i = startheavy to endheavy do
    for j = startres to endres
      x'' ←Swap the resources assigned for JJ [i] and JJJ [j]
      [localmakespan[ ], temp] ← f2(x, ETC[ ][ ])
      if ( temp < Best )
        x' ← x''
        Best ← temp
      endif
    endfor
  endfor
  if (fitnes > Best) then
    fitnes ← Best
    x ← x'
    break
  endif
endfor
```

References

1. Montero, R.S.; Huedo, E.; Lorente, I.M.: Benchmarking of high throughput computing applications on grids. J. Parallel Comput. **32**, 267–279 (2006)
2. Xue, Y.; Wang, Y.; Wang, J.; Luo, Y.; Hu, Y.; Zhong, S.; Tang, J.; Cai, G.; Guan, Y.: High throughput computing for spatial information processing (HIT-SIP) system on grid platform. In: Sloot P.M.A., Hoekstra A.G., Priol T., Reinefeld A., Bubak M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 40–49. Springer, Heidelberg (2005)
3. Foster, I.; Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure, 2nd edn. Elsevier and Morgan Kaufmann, San Francisco (2004)
4. Foster, I.; Kesselman, C.; Tuecke, S.: The anatomy of the grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. **15**, 200–220 (2001)
5. Braun, T.D.; Siegel, H.J.; Beck, N.; Hensgen, D.A.; Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks on heterogeneous distributed system. J. Parallel Distrib. Comput. **61**, 810–837 (2001)
6. He, X.; Sun, X.-H.; Laszewski, G.V.: QoS guided min–min heuristic for grid task scheduling. J. Comput. Sci. Technol. **18**, 442–451 (2003)
7. Ibarra, O.H.; Ki, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM **24**, 280–289 (1977)
8. Nesmachnow, S.; Alba, E.; Cancela, H.: Scheduling in heterogeneous computing and grid environments using a parallel CHC evolutionary algorithm. Comput. Intell. **28**, 131–155 (2012)
9. Brimberg, J.; Hansen, P.; Lih, K.-W.; Mladenovi'c, N.; Breton, M.: An oil pipeline design problem. Oper. Res. **51**, 228–239 (2003)
10. Audet, C.; Brimberg, J.; Hansen, P.; Mladenovi'c, N.: Pooling problem: alternate formulation and solution methods. Manag. Sci. **50**, 761–776 (2004)
11. Costa, M.C.; Monclar, F.R.; Zrikem, M.: Variable neighborhood decomposition search for the optimization of power plant cable layout. J. Intell. Manuf. **13**, 353–365 (2005)
12. Meric, L.; Pesant, G.; Pierre, S.: Variable neighborhood search for optical routing in networks using latin routers. Ann. Télécommun./Ann. Telecommun. **59**, 261–286 (2004)
13. Loudni, S.; Boizumault, P.; David, P.: On-line resources allocation for ATM networks with rerouting. Comput. Oper. Res. **33**, 2891–2917 (2006)
14. Hansen, P.; Mladenovi'c, N.; Moreno Pérez, J.A.: Variable neighborhood search: methods and applications. 4OR A Q. J. Oper. Res. **6**, 319–360 (2008)
15. Behnamian, J.; Zandieh, M.: Earliness and tardiness minimizing on a realistic hybrid flowshop scheduling with learning effect by advanced metaheuristic. Arab. J. Sci. Eng. **38**, 1229–1242 (2013)
16. Xhafa, F.; Carretero, J.; Abraham, A.: Genetic algorithm based schedulers for grid computing systems. Int. J. Innov. Comput. Inf. Control **3**, 1–19 (2007)
17. Wang, L.; Siegel, H.; Roychowdhury, V.; Maciejewski, A.: Task matching and scheduling in heterogeneous computing environments using a genetic algorithm-based approach. J. Parallel Distrib. Comput. **47**, 8–22 (1997)
18. Zomaya, A.; Teh, Y.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans. Parallel Distrib. Syst **12**, 899–911 (2001)
19. Kolodziej, J.; Xhafa, F.: Integration of task abortion and security requirements in GA-based meta-heuristics for independent batch grid scheduling. Comput. Math. Appl. **63**, 350–364 (2012)
20. Xhafa, F.; Alba, E.: Dorronsoro, B.: Efficient batch job scheduling in grids using cellular memetic algorithms. In: Proceedings of 21st International Parallel and Distributed Processing Symposium. IEEE Press, Long Beach, CA, pp. 1–8 (2007)
21. Fidanova, S.: Simulated annealing for grid scheduling problem. In: Proceedings of IEEE JVA International Symposium on Modern Computing, Bulgaria, IEEE Computer Society, pp. 41–45 (2006)
22. Chang, R.; Chang, J.; Lin, P.: An ANT algorithm for balanced job scheduling in grids. Future Gener. Comput. Syst. **25**, 20–27 (2009)
23. Ritchie, G.; Levine, J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, pp. 178–183 (2004)
24. Liu, H.; Abraham, A.; Hassanien, A.E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst. **26**, 1336–1343 (2010)
25. Selvi, S.; Manimegalai, D.; Suruliandi, A.: Efficient job scheduling on computational grid with differential evolution algorithm. Int. J. Comput. Theory Eng. **3**, 277–281 (2011)
26. Xhafa, F.; Carretero, J.; Dorronsora, B.; Alba, E.: Tabu search algorithm for scheduling independent jobs in computational grids. Comput. Inf. J. **28**, 237–249 (2009)
27. Xhafa, F.; Carretero, J.; Alba, E.; Dorronsoro, E.: Design and evaluation of tabu search method for job scheduling in distributed environments. In: Proceedings of the 22nd International Parallel and Distributed Processing Symposium. IEEE Press, pp. 1–8 (2008)

28. Dueck, G.; Scheuer, T.: Accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J. Comput. Phys. **90**, 161–175 (1990)
29. Xu, J.; Lam, A.Y.S.; Li, V.O.K.: Chemical reaction optimization for task scheduling in grid computing. IEEE Trans. Parallel Distrib. Syst. **22**, 1624–1631 (2011)
30. Wen, Y.; Xu, H.; Yang, J.: A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. Inf. Sci. **181**, 567–581 (2011)
31. Krauter, K.; Buyya, R.; Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing. Softw. Pract. Exp. **32**, 135–164 (2002)
32. Xhafa, F.: A hybrid evolutionary heuristic for job scheduling on computational grids. Stud. Comput. Intell. **75**, 269–311 (2007)
33. Abraham, A.; Liu, H.; Zhao, M.: Particle swarm scheduling for work-flow applications in distributed computing environments. Stud. Comput. Intell. **128**, 327–342 (2008)
34. Kim, S.S.; Byeon, J.H.; Liu, H.; Abraham, A.; McLoone, S.: Optimal job scheduling in grid computing using efficient binary artificial bee colony optimization. Soft Comput. **17**, 867–882 (2014)
35. Hemamalini, M.: Review on grid task scheduling in distributed heterogeneous environment. Int. J. Comput. Appl. **40**, 24–30 (2012)
36. Kamalam, G.K.; Bhaskaran, V.M.: New enhanced heuristic min-mean scheduling algorithm for scheduling meta-tasks on heterogeneous grid environment. Eur. J. Sci. Res. **70**, 423–430 (2012)
37. Qureshi, M.B.; Dehnavi, M.M.; Min-Allah, N. et al.: Survey on grid resource allocation mechanisms. J. Grid Comput. **12**, 399–441 (2014)
38. Kolodziej, J.; Khan, S.U.: Data scheduling in data grids and data centers: a short taxonomy of problems and intelligent resolution techniques. Trans. Comput. Collect. Intell. X **7776**, 103–119 (2013)
39. Torkestani, J.A.: A new approach to the job scheduling problem in computational grids. Cluster Comput. **15**, 201–210 (2012)
40. Kolodziej, J.; Khan, S.U.: Multi-level hierarchical genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. Inf. Sci. **214**, 1–19 (2012)
41. Nesmachnow, S.; Iturriaga, S.: Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm. Int. J. Grid Utility Comput. **4**, 70–84 (2013)
42. Nesmachnow, S.; Dorronsoro, B.; Pecero, J.E.; Bouvry, P.: Energy-aware scheduling on multicore heterogeneous grid computing systems. J. Grid Comput. **11**, 653–680 (2013)
43. Pinel, F.; Dorronsoro, B.; Pecero, J.E.; Bouvry, P.; Khan, S.U.: A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. Clust. Comput. **16**, 421–433 (2013)
44. Kołodziej, J.; Khan, S.U.; Wang, L.; Byrski, A.; Min-Allah, N.; Madani, S.A.: Hierarchical genetic-based grid scheduling with energy optimization. Clust. Comput. **16**, 591–609 (2013)
45. Kołodziej, J.; Khan, S.U.; Zomaya, A.Y.: A taxonomy of evolutionary-inspired solutions for energy optimization: problems and intelligent resolution techniques. In: Kołodziej, J.; Khan, S.U.; Burczyński, T. Advances in Intelligent Modelling and Simulation: Artificial Intelligence-Based Models and Techniques in Scalable Computing, Chap. 10, Studies in Computational Intelligence, vol. 422, Springer, Berlin (2012)
46. Kołodziej, J.; Khan, S.U.; Wang, L.; Zomaya, A.Y.: Energy efficient genetic-based schedulers in computational grids. Concur. Comput. (2012). doi:10.1002/cpe.2839
47. Kolodziej, J.; Khan, S.U.; Wang, L.; Kisiel-Dorohinicki, M.; Madani, S.A.; Niewiadomska-Szynkiewicz, E.; Zomaya, A.Y.; Xu, C.-Z.: Security, energy, and performance-aware resource allocation mechanisms for computational grids. Future Gener. Comput. Syst. **31**, 77–92 (2014)
48. Lindberg, P.; Leingang, J.; Lysaker, D.; Khan, S.U.; Li, J.: Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. J. Supercomput. **59**, 323–360 (2012)
49. Arabnejad, H.; Barbosa, J.G.: Budget constrained scheduling strategies for on-line workflow applications. Comput. Sci. Appl. ICCSA 532–545 (2014)
50. Rajni Chana, I.: Bacterial foraging based hyper-heuristic for resource scheduling in grid computing. Future Gener. Comput. Syst. **29**, 751–762 (2013)
51. Manavalasundaram, V.K.; Duraiswamy, K.: Association based grid resource allocation algorithm. Eur. J. Sci. Res. **78**, 248–258 (2012)
52. Abudhagir, U.S.; Shanmugavel, S.: A novel dynamic reliability optimized resource scheduling algorithm for grid computing system. Arab. J. Sci. Eng. **39**, 7087–7096 (2014)
53. Ang, T.F.; Ling, T.C.; Phang, K.K.: Adaptive QoS scheduling in a service-oriented grid environment. Turk. J. Elect. Eng. Comput. Sci. **20**, 413–424 (2012)
54. Chitra, P.; Sruthi, R.: Load balanced scheduling of independent tasks in heterogeneous computing systems. Int. J. Inf. Technol. Commun. Converg. **2**, 187–203 (2012)
55. Zhi-jie, L.; Cun-rui, W.: Resource allocation optimization based on load forecast in computational grid. Int. J. Eng. Res. Appl. (IJERA) **2**, 1353–1358 (2012)
56. Kavitha, G.; Sankaranarayanan, V.: A novel resource selection framework to improve QoS in computational grid. J. Int. Comput. Sci. Eng. **9**, 130–138 (2014)
57. Aron, R.; Chana, I.: QoS based resource provisioning and scheduling in grids. J. Supercomput. **66**, 262–283 (2013)
58. Kołodziej, J.: Security-aware independent batch scheduling in computational grids. Evolutionary hierarchical multi-criteria meta-heuristics for scheduling in large-scale grid systems. Stud. Comput. Intell. **419**, 81–111 (2012)
59. Habibizad Navin, A.; Azari Khosroshahi, N.; Pourhaji Kazem, A.: Multi criteria trust model in grid computing systems. Int. J. Adv. Res. Comput. Sci. **4**, 55–59 (2013)
60. Aron, R.; Chana, I.: Formal QoS policy based grid resource provisioning framework. J. Grid. Comput. **10**, 249–264 (2012)
61. Navin, A.H.; Navimipour, N.J.; Rahmani, A.M.; Hosseinzadeh, M.: Expert grid: new type of grid to manage the human resources and study the effectiveness of its task scheduler. Arab. J. Sci. Eng. **39**, 6175–6188 (2014)
62. Lusa, A.; Potts, C.N.: A variable neighbourhood search algorithm for the constrained task allocation problem. J. Oper. Res. Soc. **59**, 812–822 (2008)
63. Moghaddam, K.; Khodadadi, F.; Entezari -Maleki, R.: A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment. Int. Workshop Inf. Electron. Eng. Proced. Eng. **29**, 3808–3814 (2012)
64. Schopf, J.: Ten actions when super scheduling, document of scheduling working group, global grid forum (2001). http://www.ggf.org/documents/GFD.4.pdf
65. Mateescu, G.: Quality of service on the grid via metascheduling with resource co-scheduling and co-reservation. Int. J. High Perform. Comput. Appl. **17**, 209–218 (2003)
66. Khokhar, A.A.; Prasanna, V.K.; Shaaban, M.E.; Wang, C.L.: Heterogeneous computing: challenges and opportunities. IEEE Comput. **26**, 18–27 (1993)
67. Siegel, H.J.; Dietz, H.G.; Antonio, J.K.: Software support for heterogeneous computing. ACM Comput. Surv. **28**, 237–239 (1996)
68. 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), pp. 199–206, Santa Fe, New Mexico USA, April 2004
69. 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

70. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, vol. 5, pp. 287–326. Elsevier, Amsterdam (1979)
71. Mladenovic', N.; Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–100 (1997)
72. Mladenovic', N.; Hansen, P.: An introduction to variable neighborhood search. In: MetaHeuristics: Advances and Trends in Local Search Paradigms for Optimization, Chapter 30, pp. 449–467. Kluwer Academic, Boston (1999)
73. Mladenovic', N.; Hansen, P.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–67 (2001)
74. Hansen, P.; Mladenovic', N.: An introduction to variable neighborhood search. In: Handbook of MetaHeuristics, Chapter 6, pp. 145–184. Kluwer, Amsterdam (2003)
75. Hansen, P.; Mladenovic', N.; MorenoPe' rez, J.: Developments of variable neighborhoodsearch. Ann. Oper. Res. **175**(1), 367–407 (2010)
76. Hansen, P.; Mladenovic, N.; Urosevic, D.: Variable neighborhood search and local branching. Comput. Oper. Res. **33**(10), 3034–45 (2006)
77. Xhafa, F.; Duran, B.: Parallel memetic algorithms for independent job scheduling in computational grids. In: Recent Advances in Evolutionary Computation for Combinatorial Optimization, vol. 153 of Studies in Computational Intelligence, pp. 219–239. Springer, Berlin (2008)
78. 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, vol. 4446 of Lecture Notes in Computer Science, pp. 1–12. Springer, Berlin (2007)
79. Nesmachnow, S.; Cancela, H.; Alba, E.: A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Appl. Soft Comput. **12**, 626–639 (2012)
80. Ali, S.; Siegel, H.; Maheswaran, M.; Ali, S.; Hensgen, D.: Task execution time modelling for heterogeneous computing systems. In: Proceedings of the 9th Heterogeneous Computing Workshop, p. 185. IEEE Press, Washington (2000)
81. http://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP