Hybrid Meta-Heuristic Algorithms for Independent Job Scheduling in Grid Computing

Muhanad Tahrir Younis*, Shengxiang Yang

Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, UK

Abstract

The term 'grid computing' is used to describe an infrastructure that connects geographically distributed computers and heterogeneous platforms owned by multiple organizations allowing their computational power, storage capabilities and other resources to be selected and shared. The job scheduling problem is recognized as being one of the most important and challenging issues in grid computing environments. This paper proposes two strongly coupled hybrid meta-heuristic schedulers. The first scheduler combines Ant Colony Optimisation and Variable Neighbourhood Search in which the former acts as the primary algorithm which, during its execution, calls the latter as a supporting algorithm, while the second merges the Genetic Algorithm with Variable Neighbourhood Search in the same fashion. Several experiments were carried out to analyse the performance of the proposed schedulers in terms of minimizing the makespan using well known benchmarks. The experiments show that the proposed schedulers achieved impressive results compared to other selected approaches from the bibliography.

Keywords: Hybrid meta-heuristic, Ant Colony Optimization, Genetic Algorithm, Variable Neighbourhood Search, Job Scheduling

1. Introduction

Grid Computing is an infrastructure that connects multiple heterogeneous and autonomous resources such as databases, computers and servers, from different domains – which can be geographically distributed worldwide – to perform various complex tasks. Depending upon their availability, performance and users' requirements, this infrastructure allows the dynamic sharing of various resources and thus creates a virtual supercomputer. Grid computing was mainly developed to fulfil the significantly increasing requirements for high computing power being demanded by various organizations and the scientific computing community. From this perspective, it has been used as a utility for diverse applications that require intensive computing power from commercial and non-commercial clients [1].

*corresponding author
Email address: muhanad.younis@dmu.ac.uk (Muhanad Tahrir Younis)

The early definitions of grid computing systems were introduced in [2] [3], and since then many developments have taken place in terms of both in grid infrastructure and middleware, resulting in a better understanding of grid issues. One of the main challenges in a computational grid is how to efficiently map jobs, also called tasks or applications, to grid resources and hence utilize geographically distributed computers which are connected through heterogeneous environments in an efficient, reliable and secure manner. This mapping is called job scheduling in grid computing. Similar to job scheduling in traditional computing systems, this mapping is known to be an NP complete problem [20]. However, it is more complicated in grid computing due to its complex, dynamic nature, high degree of job and resource heterogeneity, problem size, and other factors such as existing local schedulers and policies [10].

The problem of job scheduling in grid computing has been addressed using different approaches such as simple queuing algorithms, deterministic heuristic algorithms and meta-heuristic algorithms. However, to effectively deal with its complexity, meta-heuristic algorithms are preferred [**?**]. Meta-heuristic algorithms are well-known approaches which have been applied effectively to a wide range of NP complete problems. In fact, these algorithms are considered the best candidate in practice to cope with the complexity of job scheduling in a computational grid, and accordingly several algorithms have been suggested [**?**].

Meta-heuristic approaches such as the Genetic Algorithm (GA), Tabu Search (TS), Particle Swarm Optimisation (PSO), Variable Neighbourhood Search (VNS) and Ant Colony Optimisation (ACO) have all proven their effectiveness in solving different scheduling problems. However, hybridising two or more meta-heuristics shows better performance than applying a stand-alone approach [4]. The new high level meta-heuristic will inherit the best features of the hybridised algorithms, increasing the chances of skipping away from local minima, and hence enhancing the overall performance [23].

Four main issues that must be examined in order to design a new hybrid algorithm, namely the number of algorithms to hybridize, the type of algorithms to hybridize, the execution mode (sequential or parallel) and the hybridization type [14]. Meta-heuristic methods can be hybridized in two ways, either as loosely coupled or strongly coupled [13]. The former refers to the case in which the hybridized algorithms preserve their identity by running as a chain of executions in which the output of the first algorithm will be used by the second, and so on; the final solution will be the output of the last algorithm. The latter refers to the type of hybridization in which the inner procedures of the hybridized algorithms are interchanged in such a way as one of the methods acts as the main algorithm which, during its execution, calls other methods to act as supporting algorithms [23].

The literature shows that the use of hybrid meta-heuristics, such as ACO [6], GA [10] and VNS [16], for the job scheduling problem in grid computing provided impressive results; however, almost all the proposed meta-heuristic hybrid algorithms in the literature were loosely coupled. Moreover, the VNS algorithm is considered a framework for building heuristic algoritjms which has been used efficiently and effectively in various optimization problems [26]. However, for the job scheduling problem in grid computing, VNS was used as a stand-alone algorithm in all the available works [16]. These observations pave the way to the examination of the hybridisation of ACO and GA with VNS in a strongly coupled

fashion and compare the performance of such coupling with existing methods.

In this work, the application of VNS for the job scheduling problem in grid computing is introduced. Four new neighbourhood structures, together with a modified local search, are proposed. The proposed VNS is hybridised using two meta-heuristic methods in a strongly coupled fashion, yielding two new sequential hybrid meta-heuristic algorithms for the problem of job scheduling in grid computing. The first algorithm, called ACO-VNS, combines a modified ACO of our previous work [8] and VNS in which the former acts as the primary algorithm which calls during its execution the latter as a supporting algorithm, i.e., ACO(VNS), while the second algorithm, called GA-VNS, integrates a newly proposed GA and VNS in the same manner, i.e., GA(VNS). Using the ETC simulation model, several experiments have been carried out to evaluate the performance of the proposed methods in terms of minimising the makespan. Three different well known datasets were used for this purpose rather than generating a special dataset so that we could easily make a fair comparison to other state-of-the-art methods.

The reminder of the manuscript is organized as follows. The second section presents the related work on solving the problem of job scheduling in grid computing using meta-heuristic methods. The third section describes the simulation model used to evaluate the performance of the proposed methods. The use of VNS, ACO-VNS and GA-VNS for job scheduling in grid computing are outlined in the fourth, fifth and sixth sections, respectively. The seventh section discusses parameter tuning, while the experimental results of applying the proposed methods are presented in the eighth section. Our conclusions are drawn in the ninth and final section.

2. Related work

The problem of job scheduling in distributed and heterogeneous computing environments, such as grid computing and cloud computing, has been addressed using different approaches such as simple queuing and heuristic algorithms. However, to effectively deal with the associated complexity, meta-heuristic algorithms are preferred [**?**]. Meta-heuristic algorithms are well-known approaches which have been used effectively to solve a wide range of NP complete problems. In fact, these algorithms are considered the best candidate, in practice, to cope with the complexity of job scheduling in grid computing, therefore, several algorithms have been suggested [**?**].

Ant Colony Optimization (ACO) is one of these methods. ACO is a meta-heuristic search method which simulates the behaviour of ants in foraging for food [5]. A loosely coupled hybrid ACO algorithm has been suggested in [6] which combines ACO with Tabu Search (TS) to improve the performance of a number of similar approaches proposed in [7]. Their experimental results demonstrate that the hybridization of TS with ACO improves the makespan of the solutions. However, the hybrid method needed over 3.5 hours to achieve these results.

A loosely coupled hybrid algorithm, which combines a newly proposed Ant Colony Optimisation (ACO) and the Genetic Algorithm (GA) proposed in [7], i.e. ACO+GA, was suggested in [8] as a promising algorithm. ACO starts first and the output of it will be

used by GA which further improves it. The ETC model was used to evaluate the proposed method by generating a special 512x16 dataset using the range-based method proposed in [19]. The literature also includes another loosely coupled hybrid ACO-based method proposed in [4] for job scheduling which combines ACO and GA. However, the authors have also used a non-standard dataset and their implementation is not available to make a fair comparison.

A swarm intelligence-based scheduler for processing computational mechanics applications on federated clouds was proposed in [**?**]. The proposed scheduler aimed at minimising both the makespan and the weighted flowtime. It involves scheduling at three levels, namely broker level, infrastructure level and virtual machines level. In the first level, the proposed scheduler uses three simple heuristics to select data-centres based on their network latencies. In the second level, the authors used two meta-heuristic methods, which are ACO and PSO, to select virtual machines from the chosen data-centres at the first level. In the ACO-based scheduler, an ant is initialised whenever a virtual machine is created in a data-centre. The initialised ant searchs for the best host, in terms of minimising both the makespan and the weighted flowtime, to which it can be assigned. Similarly, the PSO-based scheduler creates a bee for every virtual machine in the data-centres. Finally, the scheduler at the third level allocates jobs to the preassigned virtual machines using a job priority heuristic.

ACO is not the only approach; the literature includes many other meta-heuristic algorithms such as the Genetic Algorithm (GA). GA is a meta-heuristic search method that mimics the evolution of living beings. It has been successfully used for solving many NP-hard optimization problems such as job scheduling in heterogeneous environments and grid computing. A study presented in [7] suggested eleven static heuristic algorithms to solve the job scheduling problem in heterogeneous environments (in terms of minimising the makespan). Their experiments show that the proposed GA achieves better results than the other 10 algorithms explored in the work. The min-min heuristic [9] was used to seed one individual of the initial generation that consists of 200 solutions, while the remining 199 individuals were generated randomly.

A GA-based scheduler has been proposed in [10] for solving the multi-objective job scheduling problem in computational systems. Two *ad hoc* heuristics, which are the Minimum Completion Time (MCT) [12] and the Longest Job to Fastest Resource−Shortest Job to Fastest Resource (LJFR−SJFR) [11], were employed along with the random approach to seed the initial generation in order to introduce diversity. The direct and the permutation encoding schemes have been considered by the authors. Moreover, various genetic operators have been implemented.

A strongly coupled hybrid algorithm has been proposed in [13] for the job scheduling problem which runs GA as the main method and calls Tabu Search (TS) to enhance the quality of solutions in the population, while a hybrid method was suggested in [14] which combines the same methods in a loosely coupled fashion. The HyperSim-G Grid simulator [15] was used for evaluating both algorithms.

A GA-based scheduler for the assignment of virtual machines on federated clouds was suggested in [**?**]. The proposed scheduler aimed at minimising the total cost of the overall assignment. The scheduling process involves two steps, namely selection of servers and

selection of data-centres. In the first step, the proposed scheduler uses a GA-based algorithm to select servers in a data-centre, while the scheduler uses a shortest path method to select data-centres on the federated clouds in the second step. The authors used an extension of the CloudSim simulator, called the REALcloudSim simulator, to test the performance of the proposed scheduler.

Besides ACO and GA, the literature includes another meta-heuristic method called Variable Neighbourhood Search (VNS), which is based on the systematic change of the neighbourhood during the search process. The traditional VNS starts with an initial solution, then explores its neighbourhoods and moves to a new one if an improvement is found. The exploration step is followed by a local search in order to move from solutions in the neighbourhood to a local optimum. A Multi-objective Variable Neighbourhood Search (MVNS) algorithm has been proposed in [**?**] to minimise both makespan and flowtime. The authors have introduced five different neighbourhood structures and the random Problem Aware Local Search Heuristic (PALS) was used. The performance of the proposed method has been compared with some of the methods discussed in the literature and the results show that MVNS outperforms all of them in all of the cases tested. The work introduced by [16] studied the use of Two-Phase Variable Neighbourhood Search (TPVNS) for task scheduling on heterogeneous computing and grid systems. Six different neighbourhood structures have been introduced and random PALS was also used. The performance of the proposed method has been evaluated against a number of methods discussed in the literature and the results show that TPVNS outperforms them in most of the cases investigated.

Several other meta-heuristic methods are also available in the literature. A fuzzy Particle Swarm Optimisation (PSO) meta-heuristic algorithm for task scheduling on computational grid was suggested by [17]. Their work focused on the minimisation of the makespan time. The authors used small- to large-scale resource-job pair problems to test the associated performance. Their method was compared with a GA and a Simulating Annealing(SA) approach. The results showed that the fuzzy PSO scheduler has the ability to find faster and feasible solutions over the GA and SA. The authors in [18] developed a differential evolution (DE) algorithm to generate schedules which efficiently utilise available resources and complete the jobs in the minimum time. The results have been compared with findings in [17] and it has been found that PSO outperforms DE in three instances. However, the authors in [18] claimed that the solutions found by DE show better resource utilisation.

Despite the many hybrid meta-heuristic algorithms available in the literature which have tackled the job scheduling problem in grid computing, only one research effort [13] examined the use of a strongly coupled hybrid meta-heuristic. Moreover, although VNS, whether it has been hybridized with other algorithms or employed as a stand-alone algorithm, has proved its effectiveness in dealing with numerous complex hard optimization problems [26], the hybridization of the VNS algorithm with other meta-heuristics has not been previously studied with regard to the job scheduling problem on computational grids. Hence, there remains room to contribute to these lines of research.

3. Simulation model

To study the job scheduling problem under different types of heterogeneous environments, a model that simulates the processing time of jobs on resources is required. Since the early 2000s, a common model called the Expected Time to Compute (ETC) has been used for this purpose [29]. The ETC model, which was introduced in [19], provides a framework for testing the performances of different scheduling algorithms under various circumstances.

This model assumes that an accurate estimation or prediction of the size of each job, the computing power of each resource, and an estimation of the load of the resources are known in advance. Furthermore, an accurate estimation of the expected execution time for each job on each resource is computable or is otherwise assumed to be known beforehand. These assumptions are realistic since it is easy to collect information about the computation power of resources and the jobs requirements from specifications provided by the user, by predications or from historic data [14]; see [**?**] [**?**] [**?**] for more details about the methods used for calculating this estimation based on analytical benchmarking and job profiling. This estimation is represented in a two-dimensional array called the ETC, where row k of the ETC array consists of the estimated execution times for job k on each resource. Similarly, column p of the ETC array contains the estimated execution times of resource p for each job. Therefore, ETC[i][j] indicates the expected execution time the job i needs to finish on resource j. It also assumed the the ETC [i, j] entry may include the time required to move the job i and any data related to it from their known source to resource j [7].

The problem description under the ETC model can be formulated as:

1. A set of n independent jobs J=$\{j_0, j_1, j_2, ..., j_{n-1}\}$ to be assigned to grid resources. Any job can be handled by any resource. However, these jobs are non-preemptive, i.e., each job should be executed entirely by one resource only.
2. A set of m heterogeneous machines R=$\{r_0, r_1, r_2, ..., r_{m-1}\}$ to be used for processing the submitted n independent jobs.
3. The ETC matrix is of size n x m, where ETC[j][r] denotes the estimated required time for processing the job j by the resource r.
4. The goal of job scheduling in grid computing is to find a mapping of the submitted jobs to the available resources that minimizes the makespan, which itself represents the finishing time of the latest task and can be computed by Equation 1.

$$makespan = min_{s \in S} max_{j \in J}(Finish\_j) \tag{1}$$

where S is the set of all possible solutions and $Finish\_j$ represents the time by which the job j is finished [**?**].

The ETC matrix can be simply generated by dividing the size of a job i by the computing power of a resource j. One example of this type is the dataset of Liu *et al.* [17], publicly available from `http://dx.doi.org/10.13140/RG.2.2.10787.04649`, which consists of four instances of different sizes. The authors used the notation (the number of resources, the number of jobs) to describe each instance. The resource job pairs vary from small-scale instance (3, 13) to large-scale instances, such as (5, 100), (8, 60) and (10, 50).

However, to capture the various characteristics of grid computing environments, two methods, which are range-based and coefficient of variation (described in [19]), have been proposed to generate ETC matrices.

Each method defines three different types of metric, namely consistency, job heterogeneity, and resource heterogeneity. An ETC matrix is said to be consistent if a resource R_a can process a job J_x faster than a resource R_b, then the same is true for any job J_k. If this structure is not maintained, i.e., R_a is not always faster R_b, then the ETC matrix is considered inconsistent. A mix of these two scenarios is the semi-consistent ETC matrix, which can be defined as an inconsistent ETC matrix with a consistent sub-matrix. Job heterogeneity indicates the degree to which the job processing times vary with two values, high or low. Similarly, resource heterogeneity models the degree to which the resource processing times vary for a given job and also has two values, high or low. Therefore, twelve distinct ETC matrices are needed so that we can consider all these various characteristics.

Both methods follow the same design structure to generate ETC instances. Although the coefficient of variation method allows for greater control over job and resource heterogeneity through the use of empirical probability distributions, it uses a more complex procedure than the range-based method [29]. Two ranges are used by the range-based procedure to generate ETC problem instances, namely [1, R_{job}] and [1, $R_{resource}$] for job and resource heterogeneity, respectively. For every job x, the method first generates a random number, Job, by sampling a uniform distribution from the first range. Similarly, the method generates another random number, Res, from the second range for every resource y. Then, the rows of ETC matrix are constructed by multiplying Job by Res [19]. The main steps to generate ETC problem instances using the range-based method are listed in algorithm 1.

Algorithm 1 The range-based procedure for generating ETC instances.

```
1: let n and m be the number of jobs and resources respectively.
2: for (x = 0 to n-1) do
3:     Job ← a uniformly distributed random number in the range [1, R_job].
4:     for (y=0 to m-1) do
5:         Res ← a uniformly distributed random number in the range [1, R_resource].
6:         ETC[x, y] ← Job * Res.
7:     end for
8: end for
```

For realistic heterogeneous computing systems such as computational grids, the authors in [19] suggested typical values for R_{job} and $R_{resource}$ which are reported in the first row of table 1. However, their work did not create a specific benchmark. The authors in [7] used the range-based method of [19] to generate the 12 classic ETC problem instances, which are publicly available from `https://www.fing.edu.uy`.

These instances, which are known as the Braun *et al.* dataset, have become a *de facto* standard benchmark to evaluate the performance of various scheduling approaches in heterogeneous environments and grid systems. However, they did not use the suggested values for R_{job} and $R_{resource}$ in [19]. Instead, the authors used the values listed in the second row of

Table 1: Job and resource heterogeneity parameters.

	job heterogeneity		resource heterogeneity	
ETC model	low	high	low	high
Ali *et al.*	R_{job}=10	R_{job}=100000	$R_{resource}$=10	$R_{resource}$=1000
Braun *et al.*	R_{job}=100	R_{job}=3000	$R_{resource}$=10	$R_{resource}$=1000

table 1. Each instance has 512 jobs and 16 resources. The following abbreviation has been used to identify the type of ETC matrix, D-T-JHRH.0, where:

- D denotes the probability distribution type.
- T denotes the consistency type, with the following acronyms: c for consistent, i for inconsistent, and s for semi-consistent.
- JH denotes the heterogeneity of the jobs, with two possibilities either hi for high or lo for low.
- RH denotes the heterogeneity of the resources, with two possibilities either hi for high or lo for low.

The authors in [29] have reviewed the existing benchmarks in the literature. They came to the conclusion that none of the available datasets can actually simulate the current characteristics of grid computing systems in terms of dataset size. Therefore, they proposed a new benchmark, known as the Nesmachnow *et al.* dataset, which is publicly available from `https://www.fing.edu.uy`, to cover larger cases. Each case consists of 24 instances which have been generated using the range-based method described in Ali *et al.* [19]. However, the first 12 instances are generated using the job and resource heterogeneity parameters described in Ali *et al.* [19] which are listed in the second row of table 1; the other 12 are generated using the parameters proposed by Braun *et al.* [7] which appear in the first row of table 1. The notation M-d-t-jhrh was used to describe each instance where M indicates the parameters used to generate the instance. The first 12 problem instances were generated using the proposed values of Ali *et al.* [19] and therefore the letter A is used to denote them, while the parameters employed by Braun *et al.* [7] were used to generate the second twelve instances and hence the symbol B is used to represent them. The d-t-jhrh notation follows the above description.

In this work, all the three datasets, namely Liu *et al.*, Braun *et al.* and Nesmachnow *et al.*, will be considered to test the performance of the proposed methods.

4. The application of VNS to the job scheduling problem

Variable Neighbourhood Search (VNS) is a simple and effective meta-heuristic algorithm proposed by Mladenovi and Hansen in 1997 [25]. It represents a flexible framework for building heuristics for solving a set of optimization problems. VNS uses multiple neighbourhood structures to explore various neighbourhoods of the current incumbent solution, and

then selects the one which makes an improvement. The main idea of VNS is based on the systematic change of these structures both in the descent phase, in which the algorithm tries to find a local minimum, and the perturbation phase, in which VNS tries to escape from the local minimum.

VNS consists mainly of the following three steps, which are repeated until some stopping conditions are become true: the shake step, the improvement step and the neighbourhood change step. The goal of the shaking step is to resolve local minima traps by applying a set of operators in a certain order. Each operator modifies a given solution S using predefined neighbourhood structures. The improvement step involves applying a local search as an attempt to improve the solution produced by the shaking step. Two common search strategies are used within the improvement step which are the first improvement and the best improvement. The former strategy stops the local search procedure as soon as an improvement to the current solution is encountered, while the latter checks all possible solutions and selects the best among them. Beside these strategies, it is also possible to use meta-heuristic algorithms for the local search. The neighbourhood change step, which is the final step in the basic VNS procedure, is used to make a decision about the neighbourhood to be explored next and whether to accept the current solution as a new incumbent solution or otherwise. Various neighbourhood change procedures are available in the literature such as the sequential, cyclic and pipe neighbourhood change procedures [26]. The basic VNS procedure is illustrated in Algorithm 2.

Algorithm 2 The basic VNS procedure

```
1: Let: S_0 ←initial solution.
2: Let: N_k ←the set of neighbourhood structures, k ∈ [1, k_max].
3: repeat
4:     k ← 1.
5:     repeat
6:         S_1 ← shake(S_0).
7:         S_2 ← local_search(S_1).
8:         if (fitness(S_2) < fitness(S_0)) then
9:             S_0 ← S_2.
10:            k ← 1.
11:        else
12:            k ← k + 1.
13:        end if
14:    until (k==k_max)
15: until (termination condition)
```

4.1. Neighbourhood structures for job scheduling in grid computing

The neighbourhood structure provides a way to explore new parts in the solution space. This exploration is achieved through defining the type of modifications which could be applied to a given solution to produce new ones. The solution space can be explored in different

ways using different neighbourhoods; thus, using well-defined neighbourhood structures will certainly lead to better exploration.

In the job scheduling problem, any solution S will have at least one resource with a local makespan time equal to the overall makespan of the solution, where this resource is called the 'problem resource'. New solutions can be obtained from S by swapping a job currently assigned to the problem resource with a job assigned to other resource, or by moving a job currently assigned to the problem resource to a different resource. Therefore, we can define many new neighbourhood structures based on the concepts of swap and move. In this study, four new structures have been proposed, which are the Penalty-based Swap (PS), the Penalty-based Move (PM), the Random Max to Min Move (RMMM) and the Longest Max to Min Move (LMMTM).

The first structure, PS, alters the solution by finding the set of exchanges of some of the jobs assigned to the problem resource with some of the jobs assigned to other resources which best improve the solution in terms of minimising the makespan. The general PS procedure is illustrated in Algorithm A.1. On the other hand, PM modifies the solution by finding the set of moves which reallocates some of the job assigned to the problem resource to other resources that best reduce the makespan time, as shown in Algorithm A.2, while RMMM, the third neighbourhood structure, moves a random job from the list of the jobs assigned to the problem resource to the resource with the minimum local makespan time, as demonstrated in Algorithm A.3. Finally, LMMM defines new neighbours by moving the job in the list of jobs assigned to the problem resource which has the maximum expected completion time to the resource which has the minimum processing time for it. Algorithm A.4 describes the general steps of LMMM.

4.2. The improvement step

As we mentioned earlier, this step involves applying a local search procedure to improve the solution generated form the shaking step. In this study, a modified version of the Problem Aware Local Search (PALS) is used. PALS was originally proposed to solve the problem of DNA fragment assembly problem [27] [28], and a variant thereof called Randomized PALS has been used for job scheduling in heterogeneous environments [29] [16]. Recently, it has been used as an efficient technique for some permutation-based optimisation problems [30].

The general steps of the modified random PALS are described in Algorithm 3. The algorithm selects a resource Pres, which is the resource with the largest local makespan, and a random resource Rres such that $Pres \neq Rres$ from a given solution S.

The outer loop iterates on some of the jobs of Pres. The number of these jobs is selected according to the generation of a random number, $Pres\_start$, which belongs to the range [1, $Pres\_list\_size - 1$] where $Pres\_list\_size$ is the number of jobs assigned to Pres and another random number, $Pres\_end$, from the range $[Pres\_start, Pres\_list\_size]$.

Similarly, the inner loop works on the jobs of Rres using $Rres\_start$ and $Rres\_end$ which are generated in the same manner as $Pres\_start$ and $Pres\_end$, respectively. This process, which was suggested in [16], guarantees the selection of different jobs with different sizes in each iteration, while the randomised PALS used in [29] has used a different method in which the start job of each loop is generated randomly and the number of jobs in the outer loop is

Algorithm 3 The Modified Random Problem Aware Local Search Procedure

```
1: for (iter=1 to max_iter) do
2:     sol_makespan ← makespane(S).
3:     Ś ← S .
4:     Min_makespan ← ∞.
5:     Find the problem resource (Pres) which has the maximum local makespan time.
6:     Pres_Job_List ← job list of the resource with maximum local makespan.
7:     Pres_size ← Pres_Job_List size.
8:     Randomly select a resource Rres such that Rres ≠ Pres.
9:     Rres_Job_List ← job list of the resource with minimum local makespan.
10:    Rres_size ← Rres_Job_List size.
11:    Pres_start ← random(1, Pres_size − 1).
12:    Pres_end ← random(Pres_start, Pres_size).
13:    Rres_start ← random(1, Rres_size − 1).
14:    Rres_end ← random(Rres_start, Rres_size).
15:    for (i=Pres_start to Pres_end) do
16:        for (j=Rres_start to Rres_end) do
17:            S_1 ← swap_resources(Ś,Pres_Job_List[i],Rres_Job_List[j]).
18:            S_2 ← move_job(Ś,Pres_Job_List[i],Rres).
19:            if (makespan(S_1) < makespan(S_2)) then
20:                S̈ ← S_1.
21:            else
22:                S̈ ← S_2.
23:            end if
24:            current_makespan ← makespan(S̈).
25:            if (current_makespan < Min_makespan) then
26:                Ś ← S̈.
27:                Min_makespan ← current_makespan.
28:            end if
29:        end for
30:    end for
31:    if (Min_makespan < sol_makespan) then
32:        S ← Ś.
33:    end if
34: end for
```

fixed to 32, whereas the number of jobs in the inner is set to the number of jobs divided by 20.

The double loop calculates the makespan values when swapping and moving jobs in S1 and S2, respectively. Then it selects the solution with the minimum makespan and compares this with the best solution so far; if there is an improvement, then it will become the new best, otherwise it will continue. Therefore, this loop stores the best improvement to the

solution with respect to the makespan time obtained by applying ($Pres\_end$ X $Rres\_end$) swaps or transfers.

This solution is then compared with S to decide whether to accept or reject it. If it is better than S then the algorithm will accept it as the new S, otherwise it will reject it and continue. The process is repeated max_iter times which means that the best improvement is applied, whereas the random PALS used in [29] and [16] is applied until an improvement to the original solution is discovered or until max_iter iterations, i.e., the first improvement strategy was used.

5. The application of hybrid ACO to the job scheduling problem

ACO is a meta-heuristic search algorithm which simulates the behaviour of ants in the process of foraging for food and how they can find a path between their nest and a source of food in this process [5]. ACO has been successfully applied for many NP complete problems which are closely related to the problem of job scheduling in computational grids [31]. This section introduces an ACO-based scheduler which is a modified version of previous work [8].

Selecting a structure to represent the solution of the problem under examination is the first step in any ACO-based algorithm. A colony of k ants is used in which each ant represents the whole solution to the scheduling problem. More precisely, each ant in the colony is represented as a list with a size equal to the total number of jobs. The value of ant[a] represents the resource into which the job a is assigned. Thus, we should expect integers in the range [0, total number of resources-1] for this list.

The ACO algorithm uses two types of information to find a solution to an optimisation problem, namely the pheromone trial and the heuristic information. The ants use the pheromone trail to communicate between them. This communication involves sharing useful information about optimal solutions. A pheromone matrix, τ, of size n x m is required, where τ[j][r] represents the favourability of allocating job j to resource r. The second piece of information the ants use to construct their solutions is the heuristic function η_{jr}. In this study, the following heuristic is used, which was proposed in [**?**]:

$$\eta_{jr} = \frac{1}{free[r]} \tag{2}$$

where the function free[r] represents the time by which the resource r becomes free. η_{jr} will be a large value if free[r] is small. Thus, a resource will be more desirable if it is free earlier.

A fitness function is required to measure the quality of the solutions, which is the makespan in our case. The general throughput of the grid system can be indicated by the makespan value of the schedule; a small makespan means that the scheduler is producing a high-quality mapping that best utilizes the available resources.

After each iteration, the pheromone deposit is updated as defined in Equation 3, which follows the updating rule described in [**?**]. This update allows the indirect communication among ants to share information about the current states of the resources.

$$\tau_{jr} = \begin{cases} \rho * \tau_{jr} + \Delta\tau_{jr} & \text{if job j is assigned to resource r in local\_best\_ant} \\ \rho * \tau_{jr} & \text{otherwise} \end{cases} \tag{3}$$

where ρ, $(0 < \rho \leq 1)$, represents the decay parameter the ants use to forget poor solutions and $\Delta\tau_{jr}$ denotes the amount of pheromone deposit on the path and is defined by Equation 4 as:

$$\Delta\tau_{jr} = \frac{makespan(local\_best\_ant)}{makespan(global\_best\_ant)} \tag{4}$$

Rules 3 and 4 allow the best ant only to deposit pheromone after each iteration. The best ant could be defined as the *local_best_ant* (the best ant in the current iteration) or the *global_best_ant* (the best ant so far). It is worth mentioning that we used different rules to update the pheromone trial and for $\Delta\tau_{jr}$ in our previous work [8].

To construct its solution, each ant uses the heuristic function as well as the information encoded in the pheromone trial. Moreover, every ant maintains two lists, namely mapped and unmapped lists. The former is initially empty, while the latter contains all the submitted jobs. The ACO-based scheduler chooses the first job-resource pair randomly. The next job-resource pair is picked out probabilistically by mapping the job j to the resource r using the transition rule defined in Equation 5 as follows:

$$p_{jr} = \frac{[\tau_{jr}]^{\alpha} * [\eta_{jr}]^{\beta} * \frac{1}{ETC[j,r]}}{\sum[\tau_{jr}]^{\alpha} * [\eta_{jr}]^{\beta} * \frac{1}{ETC[j,r]}} \tag{5}$$

where τ_{jr} and η_{jr} are the existing pheromone trail and the heuristic information, respectively, and α and β are two parameters used to define the relative weights of the pheromone and the heuristic, respectively. The pheromone trail τ_{jr} provides each ant with information about the favourability of assigning job j to resource r. On the other hand, the heuristic function η_{jr} will find the best available resource r, in terms of being free earlier, to process the job j from the unmapped jobs list. Furthermore, the inverse of the ETC[j,r] is used as an additional heuristic function; the inverse was used since lower values are more preferable.

The same steps are repeated until the unmapped list becomes empty, i.e., a solution is constructed. The same procedure is followed by every ant in the colony. When all k ants construct their solutions, the best local ant in the colony is determined and the VNS algorithm is performed on it before updating the pheromone trail. The VNS improves the solution found by the local best ant and hence there is a good possibility that the local best ant will be the next global best ant. The pheromone update rule, defined in Equation 3, is then applied. The pseudo-code of the proposed ACO-VNS scheduler is illustrated in Algorithm 4.

It is worth mentioning that the pheromone trail update rule is also applied at the beginning before starting the main ACO-VNS procedure, where the solution found by the deterministic heuristic min-min algorithm [9] is used to update the pheromone trail in order to speed up the process of finding good solutions. The min-min heuristic begins by calculating the minimum completing time (CT) for all jobs and resources. It then determines the job j with the minimum CT and assigns it to the resource that obtains it. After allocating the job j, the CT matrix is updated. The same steps are repeated until all jobs are assigned.The pseudo-code for the min-min heuristic is illustrated in Algorithm A.5.

Algorithm 4 The hybrid ACO-VNS algorithm

let num and res be the total number of jobs and resources respectively.
Set the pheromone trial τ_{nm} to a small value.
Initialise $free[0..res-1]$ to 0.
Initialise the pheromone evaporation ρ.
Initialise *global_best_ant* to the solution found by *min_min* algorithm.
$ms \leftarrow$ the makespan of *global_best_ant* using Equation 1.
$\Delta\tau_{nm} \leftarrow \frac{1}{ms}$.
Use Equation 3 to update the pheromone trail.
while (the stopping condition is not true) **do**
    for (every ant) **do**
        Randomly select the job-resource pair (a, b).
        Add (a, b) to the mapped list.
        for (all unmapped jobs) **do**
            $free[b] \leftarrow free[b] + ETC[a, b]$.
            Use Equation 2 to compute the heuristic function.
            Use Equation 5 to compute the probability matrix.
            Find the highest ρ_{wv} value.
            Determine the next job-resource pair (a=w, b=v).
            Append (a, b) to the mapped list.
        end for
    end for
    Use Equation 1 to compute the makespan to every ant.
    Find *local_best_ant*, which is the one with the minimum makespan.
    Apply VNS algorithm, i.e., *local_best_ant* $\leftarrow$ VNS(*local_best_ant*);
    if ((makespan(*local_best_ant*) $<$ makespan(*global_best_ant*)) **then**
        global_best_ant $\leftarrow$ *local_best_ant*
    end if
    Use Equation 4 to compute $\Delta\tau_{nm}$.
    Use Equation 3 to update the pheromone trail.
end while

6. The application of hybrid GA to the job scheduling problem

A Genetic Algorithm (GA) is a meta-heuristic search algorithm that mimics the natural selection process of biological evolution. The GA works on a group of solutions (individuals), called the population rather than on one solution only. It first generates an initial population randomly. Each solution in the population is evaluated using a fitness function which assigns a score that indicates the solution's quality. The GA then evolves toward an optimal solution after a number of generations through applying the genetic operators, namely selection, crossover and mutation.

The GA has been applied successfully for various complex optimization problems that share the same characteristics with the job scheduling problem in grid computing [34]. This

Algorithm 5 The hybrid GA-VNS algorithm

1: $t \leftarrow 0$
2: Generate the initial generation Gen(t) of k individuals, where Gen(t)[0]= $min\_min()$ and the remaining individuals,Gen(t)[1] to Gen(t)[k-1], are generated randomly.
3: Evaluate the fitness of each individual in the initial generation, i.e., compute Fitness(Gen(t))
4: **while** (the end criterion is not true) **do**
5: $t \leftarrow t + 1$
6: Select Parent(t) from Gen(t-1)
7: With probability p_c, recombine individuals in Parent(t) to produce Offspr1(t)
8: With probability p_m, mutate individuals in Offspr1(t) to produce Offspr2(t) using VNS algorithm.
9: Evaluate the fitness of each individual, i.e., compute Fitness(Offspr2(t))
10: Replace Gen(t) from Offspr2(t) and/or Gen(t-1)
11: **end while**
12: return Best found solution

section introduces the use of a strongly coupled hybrid GA for the job scheduling problem in grid computing. Algorithm 5 illustrates the pseudo-code of the proposed hybrid GA-VNS and the following subsections explain the main parts of it.

6.1. The solution representation

A key issue in genetic algorithm is the representation of individuals. Two types of encodings have been reported in the literature: the direct representation and the permutation-based representation [10]. In this study, we will consider the direct representation only.

Each solution is represented as a list in the direct representation. The list size is equal to the total number of jobs. The individual[a] represents the resource into which the job a is assigned. Hence, we should expect integers in the range [0, total number of resources-1] for this list.

In the permutation-based scheme, each solution is also represented as a list. However, the list size is equal to the total number of resources. The individual[a] represents the resource where a list of jobs will be assigned. Hence, we should expect integers in these lists in the range [0, total number of jobs-1]. Unlike the direct representation where genes hold the resources which can be repeated, in this representation each gene represents a list of jobs which have been assigned to it and are unique.

6.2. The initial generation

The random method is a common way to construct the initial generation of the genetic algorithm. However, several studies have showed that seeding the initial population of a genetic algorithm with solutions from other heuristic methods will introduce greater diversity and hence produce better solutions [8]. In this study, the initial population is generated as follows: one individual will be seeded with the solution found by the *ad hoc* min-min

heuristic algorithm [9], which is described in section 5. The remaining solutions are generated randomly.

6.3. The fitness evaluation

As mentioned earlier, this work will focus on minimising the makespan. Therefore, the fitness of solutions is evaluated using Equation 1. See Section 3 (Simulation model).

6.4. The selection operator

The selection operator refers to the process that determines which individuals are to be continued and allowed to reproduce and which ones deserve to be eliminated. Several selection techniques are available in the literature. In this study, the N-Tournament method suggested in [10] [34] has been used with N=4. In tournament selection, several tournaments are run among a few individuals which have been selected randomly from the population. The winner of each tournament (the one with the best fitness) is selected for the next stage.

6.5. The crossover operator

The crossover operator is equivalent to reproduction and biological crossover. New solutions (offspring) are generated by selecting individuals from the parental generation and exchanging their genes. Crossover enables the search process to explore new regions of the solution space which have not yet been explored and provide the next generation with good quality individuals. Several types of crossover operators exist in the evolutionary computing literature, which mainly depend on the solution representation. Therefore, in our case, three crossover operators, which are one-point (1P), two-point (2P) and Half Uniform Crossover(HUX), will be considered for the direct representation.

Given two parent solutions, the one-point crossover operator starts by generating a random position between 1 and the total number of jobs-1. This position serves as an exchange point which divides each parent into two parts. Two new offspring are obtained by exchanging the two first segments of the parents.

Given two parent solutions, unlike the one-point crossover, the two-points crossover operator starts by generating two random cutting points between 1 and the total number of jobs-1. These positions serve as exchange points which divide each parent into three parts. Two new offspring are obtained by exchanging the segments of the parents between the two cutting points.

Given two parent solutions, the HUX crossover exchanges exactly half of the non-matching genes (here genes represent resources according to our solution representation). To apply HUX, the Hamming distance is first calculated. The Hamming distance in our case is the number of resources that are different in the two parent solutions. This distance is divided by two. The resulting number represents the number of non-matching resources that will be exchanged between the two parents.

6.6. The mutation operator

The Mutation operator is one of the most important elements of any genetic algorithm, which is related to the exploration of the search space. By mutation, individuals are randomly altered to maintain and introduce diversity in the subsequent generations [33]. In this study, the VNS algorithm has been employed as a mutation operator. The VNS procedure is applied on each individual with a probability of p_m.

6.7. The replacement operator

The replacement operator is the process of deciding which individuals in the population need to be eliminated to make room for the new offspring. In this paper, the Steady State Strategy is used, that is, parents and offspring compete for survival, and then the best of them are selected. Although this causes a premature stagnation of the population, the use of the Steady State Strategy produces a fast convergence (minimization) of the objective function [**?**] [10] [34]. This characteristic serves our goals since we are interested in minimizing the makespan in a relatively short time.

7. Parameter Tuning

In order to perform parameter tuning, a fix set of parameters was selected from the literature for each of the proposed algorithms. The parameter tuning experiments were carried out using a number of instances with diverse characteristics from all datasets. For the proposed VNS, the examined parameter was the neighbourhood structures order only. Population size, α, β, pheromone evaporation rate (p) were among the tested parameters for the proposed hybrid ACO-VNS. On the other hand, the following parameters were examined for the proposed hybrid GA-VNS: population size, crossover type, crossover probability and mutation probability. To select the best parameter values, each algorithm was executed 30 times for each ETC instance and for each parameter, and their average was reported.

7.1. Parameter Tuning for VNS

One of the main advantages of VNS is that it does not need many parameters. The stopping condition is the maximum number of iterations, which was set to 5. As mentioned earlier, the order of neighbourhood structures will be the main parameter that will be examined, as the forward VNS version is used in this study which means that VNS starts with k=1 and then increases k by one if no improvement is found, otherwise set k=1. Since we have four different structures, we then have 24 possible combinations. Tables B.1, B.2 and B.3 illustrate the effects of changing the order of neighbourhood structures based on different instances with different characteristics and sizes. The tables show that case 24 was the best order recorded in almost all the tested cases.

7.2. Parameter Tuning for ACO-VNS

Three parameters have been examined for the hybrid ACO-VNS which are the population size, the values of α and β and the value of the pheromone evaporation (p). The population size is set to 2 due to the attempt to reduce the computational time needed to construct

solutions by ants and increase the number of generations. Various studies have suggested optimal values for α and β which vary between 1 and 10, while the suggested values for p were between 0.5 and 0.7 [6] [8] [**?**] [**?**]. Therefore, three values have been used for α and β, which are 1, 5 and 10. The results indicate that α=10 and β=1 represented the best combination, as shown in Table 2; similarly, two values were used for the pheromone evaporation (p), which are 0.5 and 0.7. The best makespan values were achieved when using p=0.7.

Table 2: Parameter tuning for ACO-VNS algorithm: α and β. The best average makespan results are reported in bold.

Case	α	β	**512x16**		
			u_c_hihi.0	**u_i_lohi.0**	**u_s_lolo.0**
1	1	1	7740919.96	105896.57	3728.38
2	1	5	7751098.51	106078.23	3733.52
3	1	10	7892111.31	106829.59	3737.49
4	5	1	7565030.42	103716.63	**3576.45**
5	5	5	7653345.35	105214.28	3625.24
6	5	10	7703753.35	105476.08	3629.75
7	10	1	**7519709.81**	**103570.61**	3576.83
8	10	5	7588542.95	103586.01	3583.05
9	10	10	7612251.74	104996.28	3601.12

7.3. Parameter Tuning for GA-VNS

Four parameters were tested for the hybrid GA-VNS algorithm which included population size, crossover type, crossover probability, mutation probability. The candidate values for the population size were 10, 20 and 30 individuals. The best results were indicated when using a population size of 20 solutions. The experiments showed a very slow improvement rate and that a greater computational time was required to find a good mapping of jobs to resources when increasing the number of individuals from 20 to 30, suggesting that using a large population size is not beneficial for the GA-VNS. As mentioned earlier, three different crossover operators were used, which were one-point crossover (1P), two-point crossover (2P) and half uniform crossover (HUX) with the best results being achieved when using the two-point crossover operator, as illustrated in Fig. 1, while Table 3 reports the values of other parameters used to compare the performance of different crossover operators. Finally, the probability of crossover and mutation were examined. A considerable number of studies in the literature suggested high crossover and mutation probabilities [10] [23] [14] [34][**?**]; therefore, the candidate values used were 0.7, 0.8 and 0.9. The best result was recorded when using pc = 0.7 and pm = 0.8, as shown in Fig. 2.

8. Experimental Results

This section discusses the experimental results of applying the two proposed hybrid metaheuristics for job scheduling in grid computing. The Java language was used to implement

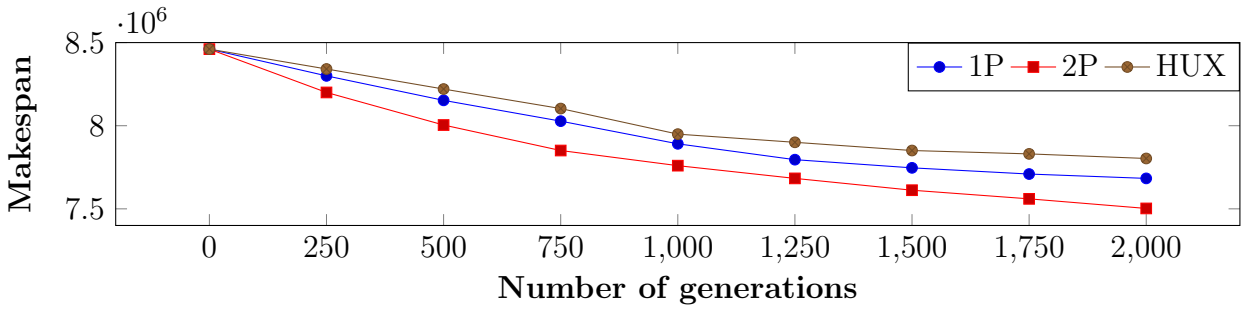

Figure 1: Parameter tuning for different crossover operators of GA-VNS using u-c-hihi.0 instance from the 512x16 dataset.

Table 3: Parameter values used for comparing the performance of different crossover operators

Seeding method	min-min algorithm
Number of generations	2000
Probability of crossover	0.7
Population size	20
Selection operator	N-Tournament,N=4
Mutation operator	VNS
Probability of mutation	0.8
Replacement operator	Steady state

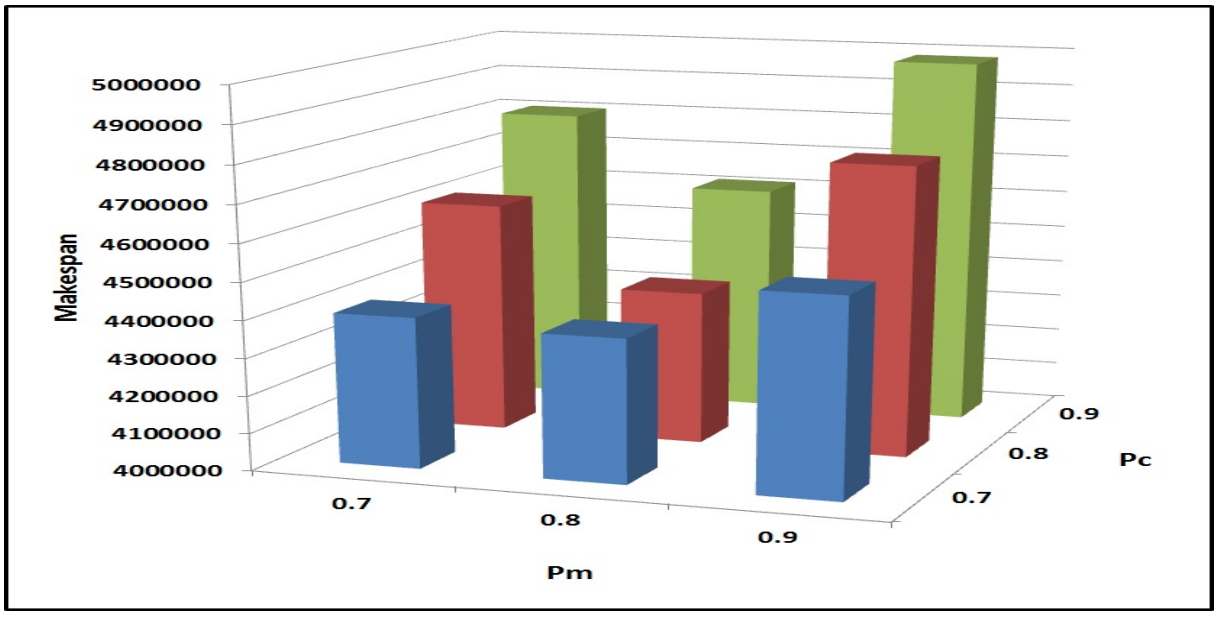

Figure 2: Analysis of GA-VNS operators probabilities using u-s-hihi.0 instance from the 512x16 dataset.

the proposed methods in this research. An Intel i5-4570 CPU @ 3.20 GHz PC with 8 GB RAM has been used to carry out all the experiments reported in this study.

In addition to the best, average, and standard deviations, the two samples with unequal variants t-test, which was used to test the hypothesis that two samples have equal means, was performed to statistically analyse the performance of the two proposed methods with a confidence interval of 95%. Moreover, two measures will also be used to compare the results obtained by applying the two proposed hybrid methods and some of the methods described in the literature. The first measure is the improvement percentage of one algorithm over, another which can be computed using Equation 6.

$$Improvement(\%) = \frac{Approach1 - Approach2}{Approach1} * 100\% \tag{6}$$

where Approach1 and Approach2 are the makespan values of the two different approaches. The second measure is the relative gap value of any approach with respect to the corresponding lower bound, which can be computed using Equation 7.

$$Gap = \frac{R - LB}{LB} \tag{7}$$

where R represents the makespan time (best or average) achieved by the proposed approach for the corresponding problem instance and the LB is the lower bound of the problem.

8.1. Results for instances from Liu et al. [17]

To enable a fair comparison, the two proposed methods use the same number of iterations as used in [17], which is (50 x the number of jobs x the number of resources) iterations, as a stopping condition. To obtain the best, average, standard deviation, and the processing time, each algorithm was executed 10 times for each instance which is also the same number used by the authors. The two proposed hybrid methods,ACO-VNS and GA-VNS, were compared against selected algorithms from the literature. In particular, the following algorithms were selected for comparison: min-min algorithm [9], Genetic Algorithm (GA) [17], Simulated Annealing (SA) [17], Particle Swarm Optimisation (PSO) [17], Differential Evolution algorithm (DE) [18], Two-Phase Variable Neighbourhood Search (TPVNS) [16] and Genetic Algorithm (MGA) [34]. All the competing algorithms were implemented sequentially. Moreover, all the above are stand-alone meta-heuristic algorithms with the exception of TPVNS, which is a loosely coupled hybrid meta-heuristic, and min-min, which is a deterministic heuristic method.

Table 4 provides the performance comparison between the two proposed hybrid methods and other methods from the literature in terms of makespan. In Table 4, the first column represents the algorithm applied, the second column represents the criteria used in comparison, namely Res (Result for deterministic min-min algorithm), Avg (average), time (in seconds), Best (best makespan found) and σ (standard deviation). There is no information provided about the best makespan achieved by the algorithms proposed in [17] and [18], the standard deviation of the algorithms suggested in [18] and [34], or the time the algorithm proposed in [16] needed to finish. The third, fourth, fifth, and sixth columns represent the four different instances. The best results are indicated in bold.

Table 4: Makespan results for dataset instances from Liu *et al.* [17]

Algorithm		(3, 13)	(5, 100)	(8, 60)	(10, 50)
min-min[9]	Res	56.0000	87.6693	47.8764	42.7346
	time	0.0001	0.0010	0.0010	0.0020
GA[17]	Avg	47.1167	85.7431	42.9270	38.0428
	σ	0.7700	0.6217	0.4150	0.6613
	time	302.9210	2415.9000	2263.0000	2628.1000
SA[17]	Avg	46.6000	90.7338	55.4594	41.7889
	σ	0.4856	6.3833	2.0605	8.0773
	time	332.5000	6567.8000	6094.9000	6926.4000
PSO[17]	Avg	46.2667	**84.0544**	41.9489	37.6668
	σ	0.2854	0.5030	0.6944	0.6068
	time	106.2030	1485.6000	1521.0000	1585.7000
DE[18]	Avg	46.0500	86.3600	42.4800	38.3900
	time	22.4400	1550.3227	430.0000	285.2600
TPVNS[16]	Best	**46.0000**	85.4345	41.7227	35.1586
	Avg	46.2500	85.4357	41.7412	35.2478
	σ	0.1100	0.1000	0.1200	0.1300
MGA[34]	Best	**46.0000**	85.5281	41.5808	35.1438
	Avg	**46.0000**	85.5333	41.5941	35.1613
	time	4.3787	195.1741	76.2255	70.4771
ACO-VNS	Best	**46.0000**	85.5281	41.5853	35.1447
	Avg	**46.0000**	85.5283	41.5918	35.1618
	σ	0.0000	0.0002	0.0077	0.0215
	time	7.2638	751.2500	392.7000	381.2500
GA-VNS	Best	**46.0000**	85.5279	**41.5795**	**35.1365**
	Avg	**46.0000**	85.5281	**41.5803**	**35.1382**
	σ	0.0000	0.0002	0.0011	0.0022
	time	4.3340	140.6454	76.0624	70.3302

The results in Table 4 show clearly that the proposed GA-VNS outperforms the other approaches, including ACO-VNS, in three instances, namely (3, 13), (8, 60) and (10, 50), while PSO [26] outperforms the other methods in the (5, 100) instance. It also shows the standard deviations of the makespans of the solutions achieved by GA-VNS are very small,

Table 5: Average improvement percentages of ACO-VNS over selected methods from the literature for dataset instances from Liu *et al.* [17].

Instance	min-min	GA	SA	PSO	DE	TPVNS	MGA
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405	0.0000
(5, 100)	2.4421	0.2505	5.7371	-1.7535	0.9631	-0.1084	0.0058
(8, 60)	13.1268	3.1105	25.0050	0.8514	2.0910	0.3580	0.0057
(10, 50)	17.7204	7.5729	15.8584	6.6503	8.4089	0.2439	-0.0017
Avg	12.7866	3.3260	11.9720	1.5812	2.8929	0.2585	0.0025

Table 6: Average improvement percentages of GA-VNS over selected methods from the literature for dataset instances from *et al.* [17].

Instance	min-min	GA	SA	PSO	DE	TPVNS	MGA
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405	0.0000
(5, 100)	2.4424	0.2508	5.7374	-1.7532	0.9633	-0.1081	0.0061
(8, 60)	13.1507	3.1371	25.0257	0.8786	2.1179	0.3854	0.0332
(10, 50)	17.7759	7.6352	15.9151	6.7132	8.4706	0.3111	0.0657
Avg	12.8065	3.3483	11.9914	1.6038	2.9151	0.2822	0.0263

Table 7: Average improvement percentages and statistical analysis of GA-VNS over ACO-VNS for dataset instances from Liu *et al.* [17].

Instance	improvement	p-value
(3, 13)	0.0000	not valid
(5, 100)	0.0003	0.01217
(8, 60)	0.0275	$<10^{-5}$
(10, 50)	0.0674	$<10^{-4}$
Avg	0.0238	

which means that the algorithm can achieve a high-quality makespan in any single execution. Furthermore, it shows the time needed to finish the search process, which clearly indicates that the proposed GA-VNS is the fastest of the meta-heuristic methods. On the other hand, the table shows the performance of ACO-VNS which showed the second-best performance after GA-VNS. However, the time need to find these results was longer than for MGA [34] and GA-VNS.

Tables 5 and 6 show the improvement percentages of ACO-VNS and GA-VNS, respectively, over the state-of-the-art methods described in the literature. Table 7 presents the improvement percentages of GA-VNS over ACO-VNS, which clearly indicates that GA-VNS performs better than ACO-VNS. Furthermore, Table 7 reports the corresponding p-value for each problem instance, all of which were less than 0.05; hence, we can reject the null hypothesis and consider the improvements in the makespan to be statistically significant. Fig. 3 shows the overall improvements of ACO-VNS and GA-VNS over the deterministic heuristic min-min algorithm for the problem instances of Liu *et al.*

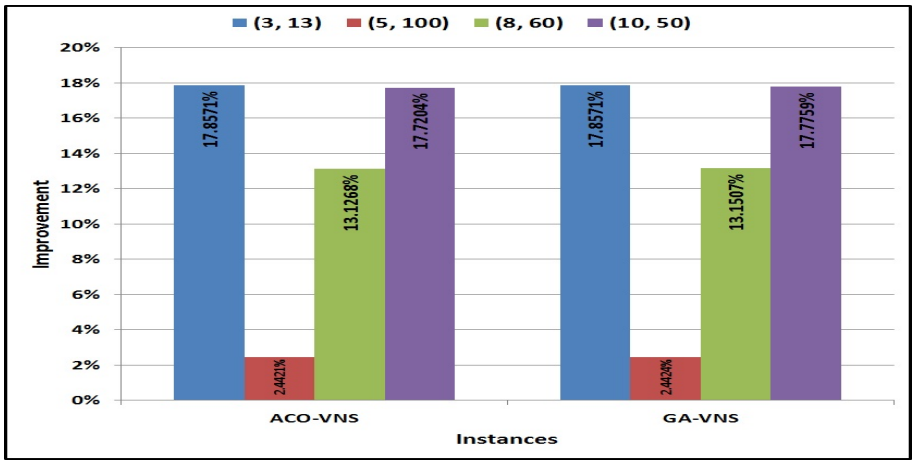

Figure 3: ACO-VNS and GA-VNS improvement percentages with respect to the min-min heuristic for Liu *et al.* dataset.

8.2. Results for instances from Braun et al. [7] 512x16

The second dataset involves the classical 12 problem instances of Braun *et al.* Each instance has 512 jobs and 16 resources. The two proposed hybrid methods, ACO-VNS and GA-VNS, were compared against selected algorithms from the literature, namely min-min algorithm [9], Genetic Algorithm (GA) [7], Cellular Memetic Algorithms (cMA) [35], Memetic Algorithm and Tabu Search (MA+TS) [24], Ant Colony Optimization and Tabu Search (ACO+TS) [6], Tabu Search (TS) [36], parallel Cross generational Heterogeneous recombination Cataclysmic mutation(pCHC) [29], and Two-Phase Variable Neighbourhood Search (TPVNS) [16]. All the competing algorithms were implemented sequentially with the exception of pCHC which was implemented using parallel mode. Moreover, all these are loosely coupled hybrid meta- heuristics apart from min-min, which is a deterministic heuristic method. Each algorithm uses different stopping times and different number of executions. For each problem instance, the deterministic min-min algorithm requires less than one second to finish the mapping, while GA, ACO-VNS, TS needed 65 s, 3.5 h, and 100 s, respectively, and the average results were reported after 100, 1, and 10 runs, respectively. cMA, MA+TS, pCHC and TPVNS required 90 s to find the solution and the average results were achieved after 10, 10, 50 and 50 runs, respectively. In this work, GA-VNS was allowed to run for 90 seconds while ACO-VNS needed 9 minutes. To obtain the best, average and standard deviation values, ACO-VNS and GA-VNS were executed 50 times for each problem instance.

Table 8 provides the results of applying ACO-VNS and GA-VNS compared to the selected methods. The best results are indicated in bold, which show clearly that GA- VNS outperforms all other approaches in all instances. The GA-VNS algorithm is expected to find high-quality schedules in any single execution since it has very small standard deviation values in the range [0.01, 0.07]. The table also shows the results for ACO- VNS which showed the second-best performance after GA-VNS with relatively small standard deviation values between 0.03 and 1.0. Although it needed six times longer than GA-VNS, it is typically 140

times faster than ACO+TS, outperforming it in 10 instances out of 12.

Tables B.4 and B.5 show the improvement percentages of ACO-VNS and GA-VNS respectively over the selected methods from the bibliography. GA-VNS shows a better improvement percentage over all the methods compared with a minimum average improvement of 0.86, while the minimum average improvement percentage of ACO-VNS, which was the second best method, was 0.30. These results indicate that ACO-VNS and GA-VNS represent the new state-of-the-art sequential hybrid algorithms for the job scheduling problem in grid computing.

The two sample t-test with unequal variants was performed to statistically analyse the performance of the two proposed hybrid methods. Table B.6 reports the improvement percentages for GA-VNS over ACO-VNS, which clearly indicate that GA-VNS performs better than ACO-VNS. Moreover, Table B.6 reports the corresponding p-value for each instance which were less than 0.05, and hence we can consider the improvement of GA-VNS over ACO-VNS in terms of makespan to be statistically significant.

In Table 8, the last column represents the Lower Bound (LB) values of each problem instance, as reported in [28]. Table 9 summarizes the gaps between the average makespan results for the proposed methods and selected algorithms from the literature and their corresponding lower bounds. GA-VNS and ACO-VNS achieved the smallest average gap values to the lower bound with 0.71 and 1.28, respectively, which indicates that the quality of the solutions they found are very high compared to the others. The average percentage gap of GA-VNS for 512x16 problem instances was 0.71%, with 8 out of 12 instances being below 1%, while ACO-VNS achieved an average gap percentage of 1.28% with 5 out of 12 being less than 1%.

Table 9: The gap values of the average makespan for the proposed methods and selected algorithms from the literature and the corresponding lower bounds for 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS	ACO-VNS	GA-VNS
u_c_hihi.0	15.17	9.59	4.82	2.50	2.05	1.39	1.83	2.01	1.45	0.78
u_c_hilo.0	7.41	2.32	1.73	0.80	1.00	0.37	0.80	1.11	0.59	0.28
u_c_lohi.0	15.83	8.66	5.55	3.00	2.50	1.48	2.23	2.48	0.93	0.48
u_c_lolo.0	8.06	2.72	1.66	0.80	0.89	0.43	0.95	1.12	0.60	0.31
u_i_hihi.0	20.78	6.72	9.53	5.13	1.32	1.67	1.64	1.60	1.05	1.03
u_i_hilo.0	10.54	3.78	3.83	2.81	0.98	0.87	1.08	1.19	0.62	0.46
u_i_lohi.0	19.25	6.37	9.46	4.70	1.37	2.77	1.60	1.52	1.37	1.09
u_i_lolo.0	9.89	3.38	3.76	2.67	0.97	0.91	1.12	1.23	0.78	0.55
u_s_hihi.0	26.99	12.37	8.88	6.34	2.44	2.59	4.32	3.31	1.88	1.01
u_s_hilo.0	9.56	3.25	3.00	1.84	1.41	0.80	1.40	0.96	0.78	0.42
u_s_lohi.0	16.46	8.44	7.94	5.96	2.88	2.45	2.94	2.91	4.32	1.40
u_s_lolo.0	13.26	4.94	3.14	2.03	1.18	1.05	1.68	1.38	1.01	0.68
Avg	14.43	6.04	5.28	3.21	1.58	1.40	1.80	1.73	1.28	0.71

8.3. Results for instances from Nesmachnow et al. [22]: 1024x32 and 2048x64

The third dataset consists of two sizes: 1024x32 and 2048x64, each of which contains 24 problem instances. Unlike the dataset of Braun *et al.* [7] , which is considered the *de facto* standard benchmark for studying the job scheduling problem in grid computing, the

Table 8: Makespan results for 512x16 dataset instances from Braun *et al.*

Instance	min-min[9]	GA[7]	cMA[35]	MA+TS[24]	ACO+TS[6]	TS[36]	pCHC[29]		TPVNS[16]		ACO-VNS			GA-VNS			LB
	res	avg	avg	avg	best	avg	best	avg	best	avg	best	avg	σ	best	avg	σ	
u_c_hihi.0	8460675.00	8050844.50	7700929.80	7530020.20	7497200.90	7448640.50	7461819.10	7481194.50	7439471.80	7494257.80	7429322.59	7452836.64	0.20%	**7391226.27**	**7403808.91**	0.07%	7346524.20
u_c_hilo.0	164022.44	156249.20	155334.80	153917.20	154234.60	153263.30	153791.90	153924.00	153270.10	154400.30	153183.78	153597.87	0.11%	**153106.67**	**153123.67**	0.01%	152700.40
u_c_lohi.0	275837.34	258756.80	251360.20	245288.90	244097.30	241672.70	241524.00	243446.30	240803.30	244043.20	239259.98	240342.04	0.20%	**239258.20**	**239289.10**	0.01%	238138.10
u_c_lolo.0	5546.26	5272.30	5218.20	5173.70	5178.40	5155.00	5177.50	5181.60	5154.80	5190.30	5148.15	5163.60	0.11%	**5148.02**	**5148.59**	0.01%	5132.80
u_i_hihi.0	3513919.25	3104762.50	3186664.70	3058474.90	2947754.10	2957854.10	2952493.20	2956905.70	2944074.60	2955764.70	2938416.02	2939907.97	0.03%	**2938380.62**	**2939301.25**	0.02%	2909326.60
u_i_hilo.0	80755.68	75816.10	75856.60	75108.50	73776.20	73692.90	73639.80	73847.10	73378.00	73927.00	73377.99	73512.45	0.05%	**73362.98**	**73391.97**	0.02%	73057.90
u_i_lohi.0	120517.71	107500.70	110620.80	105808.60	102445.80	103865.70	102136.10	102677.30	102057.50	102599.70	102052.98	102452.20	0.18%	**102051.74**	**102167.07**	0.05%	101063.40
u_i_lolo.0	2779.09	2614.40	2624.20	2596.60	2553.50	2552.10	2549.80	2557.20	2547.90	2560.10	2541.52	2548.78	0.13%	**2541.17**	**2542.98**	0.03%	2529.00
u_s_hihi.0	5160343.00	4566206.00	4424540.90	4321015.40	4162547.90	4168795.90	4198779.50	4239146.30	4145941.70	4197996.50	4106391.16	4140091.67	0.35%	**4102908.20**	**4104613.59**	0.02%	4063563.70
u_s_hilo.0	104540.73	98519.40	98283.70	97177.30	96762.00	96180.90	96623.30	96750.30	95872.30	96330.40	96117.37	96167.89	0.03%	**95788.97**	**95823.04**	0.02%	95419.00
u_s_lohi.0	140284.48	130616.50	130014.50	127633.00	123922.00	123407.40	123251.50	123989.40	122986.00	123954.30	123576.46	125650.31	1.00%	**122084.10**	**122143.20**	0.02%	120452.30
u_s_lolo.0	3867.49	3583.40	3522.10	3484.10	3455.20	3450.50	3450.10	3472.20	3440.50	3461.90	3439.35	3449.46	0.13%	**3436.63**	**3437.97**	0.01%	3414.80

literature does not include much work which addresses this dataset. The performance of ACO-VNS and GA-VNS was compared against the following algorithms from the bibliography: min-min algorithm [9], parallel Cross generational Heterogeneous recombination Cataclysmic mutation(pCHC) [29], and Two-Phase Variable Neighbourhood Search (TPVNS) [16]. All the competing algorithms were implemented sequentially apart from pCHC, which was implemented using parallel mode. Moreover, all of these are loosely coupled hybrid meta-heuristics apart from min-min, which is a deterministic heuristic method. For both sizes, GA-VNS was allowed to run for 90 seconds, which was the same time used for pCHC and TPVNS, while ACO-VNS ran for 9 minutes. To obtain the best, average and standard deviation values, both schedulers were executed 50 times for each problem instance.

Tables 10 and 11 provide the results of applying ACO-VNS and GA-VNS for the 1024x32 and 2048x64 dataset instances, respectively, compared to the selected methods. The best results are indicated in bold, which show clearly that the GA-VNS outperforms all other approaches in all instances. The GA-VNS algorithm is expected to find high quality schedules in any single execution since it has very small standard deviation values, which vary between 0.02 and 0.33 for the 1024x32 dataset, and between 0.01 and 0.26 for the 2048x64 dataset.

Table B.7 shows the percentages of makespan reduction for ACO-VNS and GA-VNS for the 1024x32 and 2048x64 dataset instances over the selected methods from the bibliography. For both sizes, GA-VNS shows a better improvement percentage over all the compared methods. ACO-VNS achieved the second-best improvement percentages for the 1024x32 dataset; however, it was only the third-best results for the 2048x64 dataset as TPVNS achieved a slightly better average improvement percentage.

In Tables 10 and 11, the last column represents the Lower Bound (LB) values of each problem instance as reported in [29]. Table 12 summarizes the gap values between the average makespan results for ACO-VNS, GA-VNS and selected algorithms from the literature and its corresponding lower bounds for the 1024x32 and 2048x64 dataset instances. For 1024x32 instances, GA-VNS achieved the smallest average gaps to the lower bound at 1.26, while min-min, pCHC and TPVNS achieved 21.00, 6.38, and 5.53, respectively. For 2048x64 instances, GA-VNS also achieved the smallest average gap with the lower bound at 2.64, while min-min, pCHC and TPVNS achieved 23.21, 6.77, and 5.46, respectively. This indicates that the quality of the solutions found by these are very high compared to the others.

Table 10: Makespan results for 1024x32 dataset instances

Instance	min-min[9]	pCHC[29]		TPVNS[16]		ACO-VNS			GA-VNS			LB
		Best	Avg.	Best	Avg.	Best	Avg.	σ	Best	Avg.	σ	
A.u_c_hihi	22508062.40	20327924.00	20510300.90	20194902.00	20284191.00	19800979.10	19818850.81	0.04%	**19602098.31**	**19636595.22**	0.07%	19449230.00
A.u_c_hilo	2255966.00	2048582.70	2058352.20	2046648.00	2050942.20	2004755.76	2033689.80	1.01%	**1957526.50**	**1959206.37**	0.02%	1951345.00
A.u_c_lohi	2155.00	1956.70	2000.00	1962.00	1970.20	1900.08	1905.64	0.15%	**1876.53**	**1880.02**	0.06%	1866.40
A.u_c_lolo	225.90	207.50	217.80	206.70	213.40	200.00	200.25	0.06%	**199.72**	**200.02**	0.08%	198.90
A.u_i_hihi	6367767.60	5169960.50	5244046.90	5167781.00	5221702.00	5100026.33	5104782.53	0.04%	**5052814.63**	**5085838.69**	0.20%	5012207.00
A.u_i_hilo	641438.40	490280.30	492699.40	489525.20	493800.10	485003.80	485316.32	0.03%	**478681.94**	**480597.47**	0.18%	474404.60
A.u_i_lohi	664.70	518.20	523.60	522.40	530.10	520.02	520.96	0.09%	**507.63**	**509.85**	0.19%	503.40
A.u_i_lolo	63.70	50.60	51.70	50.80	51.90	50.00	50.27	0.26%	**49.43**	**49.61**	0.19%	49.00
A.u_s_hihi	14125881.60	12243560.00	12439843.10	12155750.00	12306122.00	12000006.48	12011669.03	0.06%	**11690402.28**	**11730490.65**	0.11%	11553632.00
A.u_s_hilo	1319050.60	1187506.40	1214303.00	1175338.00	1185443.20	1170121.89	1175301.84	0.34%	**1137089.07**	**1143819.59**	0.17%	1126556.00
A.u_s_lohi	1380.50	1186.80	1199.20	1184.80	1194.60	1170.02	1173.05	0.13%	**1131.06**	**1135.86**	0.20%	1122.20
A.u_s_lolo	140.60	122.40	126.50	122.00	123.10	120.03	122.14	0.98%	**117.49**	**117.72**	0.17%	116.70
B.u_c_hihi	6708228.50	6169823.00	6200118.00	6189681.00	6200401.50	6081103.22	6104938.95	0.28%	**6003087.48**	**6008676.39**	0.05%	5980872.00
B.u_c_hilo	66684.50	61114.70	61390.10	60807.50	61599.20	60006.62	60341.92	0.42%	**59287.94**	**59378.59**	0.06%	58942.50
B.u_c_lohi	232011.80	215149.20	218124.80	214387.10	216481.50	210001.28	210060.30	0.02%	**209183.02**	**209701.15**	0.07%	207892.80
B.u_c_lolo	2386.30	2164.30	2208.40	2142.10	2158.30	2100.04	2103.13	0.08%	**2084.10**	**2087.69**	0.07%	2078.00
B.u_i_hihi	2164576.70	1630288.60	1670112.70	1626086.00	1628729.60	1620135.38	1625042.35	0.14%	**1586928.39**	**1601151.49**	0.31%	1567179.00
B.u_i_hilo	17083.10	15121.50	15464.10	15003.10	15715.80	15001.07	15059.49	0.16%	**14702.66**	**14848.90**	0.33%	14582.30
B.u_i_lohi	56601.20	49569.90	50128.20	49264.10	49981.20	48912.32	49431.40	0.81%	**48101.79**	**48259.18**	0.18%	47606.90
B.u_i_lolo	585.00	496.10	507.40	492.70	501.40	490.18	491.68	0.13%	**482.27**	**485.53**	0.27%	477.40
B.u_s_hihi	3967265.90	3393010.20	3430218.10	3344875.00	3392157.30	3300418.86	3328580.56	0.54%	**3217747.08**	**3248005.73**	0.33%	3178482.00
B.u_s_hilo	40691.60	35988.40	36515.60	35352.20	36911.20	35000.62	35040.84	0.06%	**34285.87**	**34705.80**	0.32%	33948.70
B.u_s_lohi	135624.60	115179.20	118070.30	114653.30	117017.10	114046.48	116387.59	1.73%	**109288.52**	**109386.28**	0.03%	108330.10
B.u_s_lolo	1333.20	1191.70	1230.30	1173.50	1196.40	1170.19	1172.33	0.10%	**1143.25**	**1153.23**	0.27%	1128.10

Table 11: Makespan results for 2048x64 dataset instances

Instance	min-min[9]	pCHC[29]		TPVNS[16]		ACO-VNS			GA-VNS			LB
		Best	Avg.	Best	Avg.	Best	Avg.	σ	Best	Avg.	σ	
A.u_c_hihi	19552221.80	18110479.10	18218285.60	17795863.00	17801492.20	17895612.14	17900436.41	0.01%	**17241145.39**	**17253426.90**	0.05%	17141977.40
A.u_c_hilo	1873134.20	1748509.20	1760141.20	1727248.00	1736971.30	1733300.10	1733703.35	0.01%	**1673538.85**	**1676784.20**	0.23%	1664592.80
A.u_c_lohi	1924.70	1798.40	1804.90	1761.00	1770.60	1761.09	1761.82	0.02%	**1708.24**	**1711.09**	0.17%	1695.30
A.u_c_lolo	191.60	177.60	178.10	174.30	175.50	176.48	176.79	0.09%	**169.32**	**169.59**	0.24%	168.30
A.u_i_hihi	3248935.40	2506258.50	2546459.70	2478011.00	2500937.20	2470227.48	2547725.05	2.36%	**2434880.15**	**2451079.04**	0.26%	2366682.10
A.u_i_hilo	365828.60	272741.30	273876.30	274378.40	276000.10	271060.28	273342.21	0.60%	**269022.07**	**269887.09**	0.19%	260904.50
A.u_i_lohi	320.90	266.30	267.50	265.90	266.20	265.10	270.54	1.56%	**262.02**	**263.50**	0.19%	255.20
A.u_i_lolo	32.30	26.40	26.50	26.70	26.90	26.01	26.76	1.93%	**25.70**	**26.00**	0.24%	25.10
A.u_s_hihi	11245679.60	9756499.70	9821934.50	9524603.00	9601364.00	9310636.32	9506570.57	1.75%	**9291522.52**	**9317604.64**	0.15%	9050260.80
A.u_s_hilo	1042948.50	924094.90	937998.80	894695.30	909381.60	880835.16	910436.53	0.66%	**874008.82**	**879869.25**	0.20%	851399.90
A.u_s_lohi	1056.00	947.10	952.30	931.60	936.10	947.85	948.69	0.05%	**906.95**	**909.99**	0.17%	888.90
A.u_s_lolo	115.30	99.60	100.40	97.00	98.90	96.09	97.82	0.55%	**95.55**	**96.21**	0.21%	92.30
B.u_c_hihi	5564664.30	5290128.20	5300316.10	5209573.00	5219961.30	5183526.58	5185288.69	0.02%	**5006270.99**	**5013212.53**	0.11%	4975778.80
B.u_c_hilo	59352.80	55316.20	55343.10	53960.30	54001.50	53126.29	54414.88	0.13%	**52506.24**	**52579.58**	0.17%	52240.60
B.u_c_lohi	190842.40	177063.40	177612.40	175429.40	176981.20	175407.88	175793.58	0.13%	**173351.66**	**173399.65**	0.01%	167381.10
B.u_c_lolo	1927.70	1814.70	1818.30	1786.30	1791.00	1785.00	1785.92	0.02%	**1780.87**	**1781.43**	0.02%	1715.00
B.u_i_hihi	929295.80	770110.60	774993.00	765966.90	769121.10	765727.53	770771.60	0.32%	**762468.55**	**764872.14**	0.18%	735101.50
B.u_i_hilo	10318.40	7906.50	7932.90	7896.90	7910.10	7896.47	7920.60	0.17%	**7882.42**	**7907.56**	0.18%	7536.30
B.u_i_lohi	34071.00	26941.20	27207.30	27118.90	27900.40	27023.34	27637.86	0.34%	**26212.04**	**26539.76**	0.20%	25681.20
B.u_i_lolo	355.70	262.40	264.70	264.90	265.80	261.13	265.42	1.19%	**255.43**	**256.77**	0.19%	250.50
B.u_s_hihi	3293157.10	2910507.60	2923857.10	2865250.00	2876310.00	2867105.20	2885341.34	0.05%	**2745557.74**	**2753099.90**	0.19%	2710024.00
B.u_s_hilo	33445.40	29442.20	29518.60	28520.40	28731.20	28441.56	29053.08	0.99%	**27912.39**	**28001.94**	0.20%	27268.00
B.u_s_lohi	111237.40	98607.00	98758.30	94777.90	95101.40	94465.30	95243.41	0.65%	**93461.51**	**93683.73**	0.20%	90727.30
B.u_s_lolo	1163.80	1014.30	1019.70	995.80	1003.20	990.35	1013.01	0.12%	**956.97**	**959.58**	0.18%	939.00

Table 12: The gaps for the average makespan for the proposed ACO-VNS and GA-VNS methods and selected algorithms from the literature and their corresponding lower bounds for the 1024x32 and 2048x64 datasets.

Instance	**1024x32**					**2048x64**				
	min-min	pCHC	TPVNS	ACO-VNS	GA-VNS	min-min	pCHC	TPVNS	ACO-VNS	GA-VNS
A.u_c_hihi	15.73	5.46	4.29	1.90	0.96	14.06	6.28	3.85	4.42	0.65
A.u_c_hilo	15.61	5.48	5.10	4.22	0.40	12.53	5.74	4.35	4.15	0.73
A.u_c_lohi	15.46	7.16	5.56	2.10	0.73	13.53	6.46	4.44	3.92	0.93
A.u_c_lolo	13.57	9.50	7.29	0.68	0.56	13.84	5.82	4.28	5.04	0.77
A.u_i_hihi	27.05	4.63	4.18	1.85	1.47	37.28	7.60	5.67	7.65	3.57
A.u_i_hilo	35.21	3.86	4.09	2.30	1.31	40.22	4.97	5.79	4.77	3.44
A.u_i_lohi	32.04	4.01	5.30	3.49	1.28	25.74	4.82	4.31	6.01	3.25
A.u_i_lolo	30.00	5.51	5.92	2.60	1.25	28.69	5.58	7.17	6.61	3.59
A.u_s_hihi	22.26	7.67	6.51	3.96	1.53	24.26	8.53	6.09	5.04	2.95
A.u_s_hilo	17.09	7.79	5.23	4.33	1.53	22.50	10.17	6.81	6.93	3.34
A.u_s_lohi	23.02	6.86	6.45	4.53	1.22	18.80	7.13	5.31	6.73	2.37
A.u_s_lolo	20.48	8.40	5.48	4.67	0.87	24.92	8.78	7.15	5.98	4.24
B.u_c_hihi	12.16	3.67	3.67	2.07	0.46	11.84	6.52	4.91	4.21	0.75
B.u_c_hilo	13.13	4.15	4.51	2.37	0.74	13.61	5.94	3.37	4.16	0.65
B.u_c_lohi	11.60	4.92	4.13	1.04	0.87	14.02	6.11	5.74	5.03	3.60
B.u_c_lolo	14.84	6.28	3.86	1.21	0.47	12.40	6.02	4.43	4.14	3.87
B.u_i_hihi	38.12	6.57	3.93	3.69	2.17	26.42	5.43	4.63	4.85	4.05
B.u_i_hilo	17.15	6.05	7.77	3.27	1.83	36.92	5.26	4.96	5.10	4.93
B.u_i_lohi	18.89	5.30	4.99	3.83	1.37	32.67	5.94	8.64	7.62	3.34
B.u_i_lolo	22.54	6.28	5.03	2.99	1.70	42.00	5.67	6.11	5.95	2.50
B.u_s_hihi	24.82	7.92	6.72	4.72	2.19	21.52	7.89	6.14	6.47	1.59
B.u_s_hilo	19.86	7.56	8.73	3.22	2.23	22.65	8.25	5.37	6.55	2.69
B.u_s_lohi	25.20	8.99	8.02	7.44	0.97	22.61	8.85	4.82	4.98	3.26
B.u_s_lolo	18.18	9.06	6.05	3.92	2.23	23.94	8.59	6.84	7.88	2.19
Avg	21.00	6.38	5.53	3.18	1.26	23.21	6.77	5.46	5.59	2.64

Table B.8 presents the improvement percentages for GA-VNS over ACO-VNS for the 1024x32 and 2048x64 dataset instances, which clearly indicate that GA-VNS performs better than ACO-VNS. Moreover, the table reports the corresponding p-value when applying the two sample t-test with unequal variants for each instance to statistically analyse the performance of the two proposed hybrid methods. The p-values were less than 0.05, and hence, we can consider the improvement of GA-VNS over ACO-VNS in terms of makespan to be statistically significant.

8.4. Results summary for Braun et al. [7] and Nesmachnow et al. [22] datasets

Table 13 and Fig. 4 summarize the average improvements of ACO-VNS and GA-VNS over the deterministic heuristic min-min algorithm for the problem instances of Braun *et al.* and Nesmachnow *et al.*, where the improvement percentages are categorized based on consistency. The average improvement percentages of ACO-VNS with respect to the min-min heuristic were always above 7% and 12% for the consistent and semi-consistent instances, respectively, while GA- VNS was more accurate showing average improvement percentages of no less than 9% and 13%, respectively. For the inconsistent instances, both algorithms achieved average improvements above 12% for the 512x16 dataset. However, this percentage increased significantly to greater than 21% for larger problem instances.

Table 14 reports the average gap percentage of ACO-VNS and GA-VNS with regard to the lower bound for the datasets of Braun *et al.* and Nesmachnow *et al.* based on the

Table 13: ACO-VNS and GA-VNS average improvement percentages with respect to the min-min heuristic based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

Consistency	512x16		A-1024x32		B-1024x32		A-2048x64		B-2048x64	
	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS
consistent	9.51	9.89	11.18	12.53	9.96	10.88	8.02	11.21	7.59	9.51
inconsistent	12.15	12.30	21.72	22.65	16.35	17.71	19.93	22.04	21.14	22.74
semiconsistent	12.26	13.21	13.50	16.06	14.06	16.42	13.37	15.79	13.21	16.50

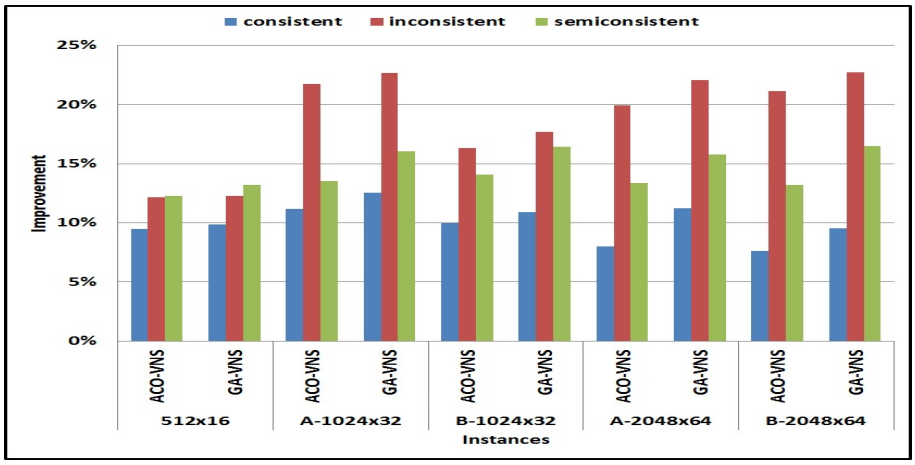

Figure 4: The graphical average improvement percentages of ACO-VNS and GA-VNS with respect to the min-min heuristic based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

consistency. For consistent and inconsistent instances, ACO-VNS showed stable behaviour. However, for the semi-consistent instances, it may be noted that ACO-VNS has a high average gap percentages for all datasets. GA-VNS showed stable behaviour regarding all types of consistency.

Fig. 5 demonstrates the average improvement percentages of ACO-VNS and GA-VNS over the *ad hoc* min-min method and the average percentages of the corresponding gap to the lower bounds of the Braun *et al.* and Nesmachnow *et al.* datasets. It may be noted that ACO-VNS and GA-VNS are capable of achieving high-quality mappings which are very close to the lower bounds. However, as the dataset size grows, the average gap percentage for ACO-VNS increases compared to GA-VNS, indicating greater stability.

Table 14: ACO-VNS and GA-VNS average gap percentages with respect to the lower bound based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

Consistency	512x16		A-1024x32		B-1024x32		A-2048x64		B-2048x64	
	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS	ACO-VNS	GA-VNS
consistent	0.89	0.46	2.22	0.76	1.68	0.64	4.39	0.71	4.38	2.22
inconsistent	0.96	0.78	2.56	1.36	3.45	1.77	6.26	3.58	5.88	3.71
semiconsistent	2.00	0.88	4.37	1.20	4.82	1.91	6.17	3.60	6.47	2.43

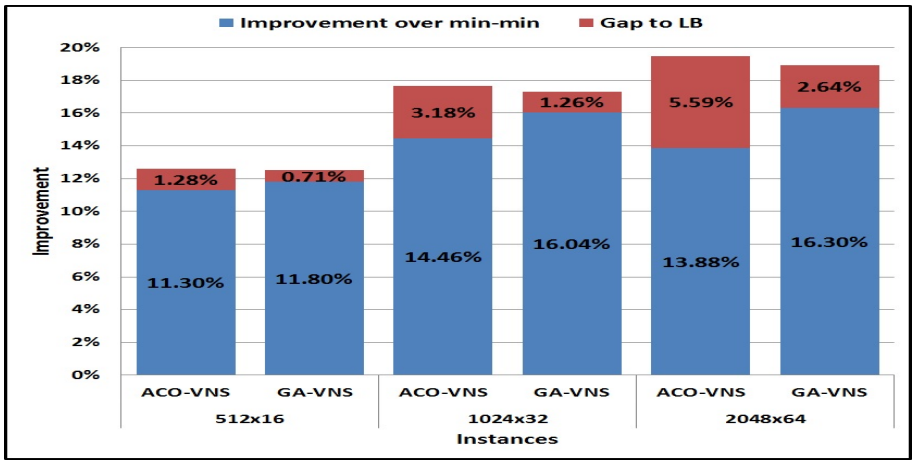

Figure 5: ACO-VNS and GA-VNS average improvement percentages over the min-min heuristic and the corresponding gap to LB: Braun *et al.* and Nesmachnow *et al.* datasets.

9. Conclusions and future work

The mapping of jobs to resources or job scheduling in distributed and heterogeneous environments such as grid computing systems is considered one of the most significant and difficult tasks. The overall performance of such systems can be improved significantly by using an effective job scheduler. The job scheduling in grid computing shares the property of being an NP complete problem with conventional distributed systems. However, in the former systems, it is particularly complex as it is dynamic, multi-objective and has a high degree of heterogeneity in terms of jobs and resources. Therefore, and to cope in practice with its difficulty and complexity, the use of meta-heuristics is necessary. ACO and GA are robust search methods which have been used to successfully solve this problem. However, the results achieved by these methods could be further improved by combining them with other meta-heuristic approaches. In this work, two meta-heuristic methods, ACO and GA, have been hybridized with a novel VNS in a strongly coupled fashion to tackle the static independent job scheduling problem in grid computing. The new high-level algorithms inherit the best characteristics of the combined methods. Four new neighbourhood structures and a modified PALS have been proposed for the novel VNS, which use the concepts of move and transfer of some jobs to or from the problem resource, which is the resource that has a local makespan equal to the total makespan of the solution. Through the use of these structures and the modified local search, VNS improves the performance of the ACO and GA algorithms by introducing diversity to the colony and the population, respectively, and by exploring new parts of the state space of the problem.

To evaluate the performance of the proposed methods, the ETC model has been used. Three different well-known datasets have been used to perform several experiments. The experimental results show that GA-VNS achieved results that were significantly better than other selected approaches from the literature for all three benchmarks used in terms of minimising the makespan; therefore, we can claim that it represents the new state-of-the-art sequential hybrid algorithm for job scheduling in grid computing. With very low standard

deviation values, it should be expected that GA-VNS can find high-quality schedules in any single run. Moreover, GA-VNS achieved results that show the smallest gap with the lower bound in all the problem instances examined in this study. ACO-VNS was almost the second-best algorithm in terms of makespan results; however, it needed a longer time to construct high-quality solutions. For relatively small problem instances, the result for ACO-VNS were very close to the ones achieved by GA-VNS; however, as the dataset size increased, the quality of the solutions found by ACO-VNS decreases, which means that longer times will be needed to improve the results.

Although the proposed methods seem promising approaches to scheduling in grid computing systems, the work presented in this line of research can be extended in various directions. We are in the process of investigating the proposed methods in a loosely coupled fashion to directly compare which hybridization scheme is better. Furthermore, research into solving the dynamic version of the problem is already underway. The suggested methods in this work addressed the minimisation of a single objective, which is the makespan. Adding another objective, such as flowtime, will convert the problem into a multi-objective one. There- fore, future work needs to be carried out to establish whether the proposed methods can tackle the multi-objective job scheduling problem in grid computing or otherwise. Moreover, the two methods proposed in this study are sequential; it would be interesting to examine their performances in parallel mode. More broadly, the prospect of being able to apply the proposed methods in this work to other distributed and heterogeneous environments, such as cloud computing systems, serves as a continuous incentive for future research.

Appendix A. Algorithms

Algorithm A.3 The Random Max to Min Move (RMMM) procedure

1: Find the problem resource (pr) which has the maximum local makespan time.
2: $Pres\_Job\_List \leftarrow$ job list of pr.
3: $Pres\_size \leftarrow Pres\_Job\_List$ size.
4: Find the resource (mr) which has the minimum local makespan time.
5: i $\leftarrow$ Random number in the range [0..$Pres\_size$]
6: S $\leftarrow$ $move\_job$(S,$Pres\_Job\_List[i]$,mr).

Algorithm A.4 The Longest Max to Min Move (LMMM) procedure

1: Find the problem resource (pr) which has the maximum local makespan time.
2: $Pres\_Job\_List \leftarrow$ job list of pr.
3: $Pres\_size \leftarrow Pres\_Job\_List$ size.
4: Find the job (lj) in $Pres\_Job\_List$ which has the longest processing time.
5: Find the resource (br) which is the fastest resource that can process lj.
6: S $\leftarrow$ $move\_job$(S,lj,br).

Algorithm A.1 The Penalty-based Swap (PS) procedure

1: Ś ← S.
2: $Min\_penalty$ ← makespan(S).
3: $penalty$ ← 0.
4: Find the problem resource (pr) which has the maximum local makespan time.
5: $Pres\_Job\_List$ ← job list of pr.
6: $Pres\_size$ ← $Pres\_Job\_List$ size.
7: **for** (i=0 to $Pres\_size - 1$) **do**
8: **for** (j=0 to total number of jobs-1) **do**
9: **if** (Ś[j] ≠ pr) **then**
10: Ś ← $swap\_resources$(Ś,$Pres\_Job\_List[i]$,j).
11: **end if**
12: $penalty$ ← makespan(Ś).
13: **if** ($penalty < Min\_penalty$) **then**
14: S ← Ś.
15: $Min\_penalty$ ← penalty.
16: **else**
17: Ś ← S.
18: **end if**
19: **end for**
20: **end for**

Algorithm A.5 The min-min algorithm

1: For every job in the job set, calculate the completion time (CT)
2: $jobs\_removed$ ← 0
3: **while** (jobs_removed<total number of jobs) **do**
4: Find the job i in the job set with the earliest completion time and the resource j which obtains it
5: Assign i to j
6: Delete i from the job set
7: $jobs\_removed$ ← $jobs\_removed + 1$
8: Update the ready time and the completion time (CT) of resource j
9: **end while**

Algorithm A.2 The Penalty-based Move (PM) procedure

Ś ← S.
$Min\_penalty$ ← makespan(S).
$penalty$ ← 0.
Find the problem resource (pr) which has the maximum local makespan time.
$Pres\_Job\_List$ ← job list of pr.
$Pres\_size$ ← $Pres\_Job\_List$ size.
for (i=0 to $Pres\_size - 1$) **do**
    for (j=0 to total number of jobs-1) **do**
        if (Ś[j] ≠ pr) **then**
            Ś ← $move\_job$(Ś,$Pres\_Job\_List[i]$,resource assigned to j).
        end if
        $penalty$ ← makespan(Ś).
        if ($penalty < Min\_penalty$) **then**
            S ← Ś.
            $Min\_penalty$ ← penalty.
        else
            Ś ← S.
        end if
    end for
end for

Appendix B. Tables

Table B.1: Neighbourhood structures order testing for 512x16 dataset using ACO-VNS and GA-VNS. The best average makespan results are reported in bold.

Case	Nieghbourhood order	ACO-VNS			GA-VNS		
		u_c_hihi.0	u_i_lohi.0	u_s_lolo.0	u_c_hihi.0	u_i_lohi.0	u_s_lolo.0
1	LMMM-RMMM-PM-PS	7523303.74	103461.03	3481.22	7401282.59	102232.20	3438.25
2	LMMM-RMMM-PS-PM	7461538.54	103078.38	3457.79	7398202.08	102179.12	3437.50
3	LMMM-PM-RMMM-PS	7611599.66	103997.77	3478.69	7411587.14	102221.50	3438.21
4	LMMM-PM-PS-RMMM	7524070.13	103482.83	3482.79	7401789.80	102201.54	3437.58
5	LMMM-PS-RMMM-PM	7600087.64	103967.27	3478.99	7410696.30	102236.81	3438.85
6	LMMM-PS-PM-RMMM	7459931.99	103198.71	3457.50	7397718.21	102120.02	3437.46
7	RMMM-LMMM-PM-PS	7551520.11	105017.89	3469.67	7405678.69	102217.07	3437.73
8	RMMM-LMMM-PS-PM	7462071.33	103116.24	3458.66	7398272.19	102159.67	3437.39
9	RMMM-PM-LMMM-PS	7599243.67	105076.61	3474.15	7410640.19	102249.57	3438.82
10	RMMM-PM-PS-LMMM	7594642.93	103994.63	3479.86	7410271.22	102193.77	3438.72
11	RMMM-PS-LMMM-PM	7553116.25	105017.03	3471.81	7406281.88	102213.49	3437.66
12	RMMM-PS-PM-LMMM	7460072.38	103203.13	3456.87	7397780.11	102106.96	3437.28
13	PM-LMMM-RMMM-PS	7591135.20	104000.94	3479.76	7410258.33	102223.43	3437.62
14	PM-LMMM-PS-RMMM	7561004.72	104996.28	3472.56	7407923.88	102219.43	3438.65
15	PM-RMMM-LMMM-PS	7589684.18	104002.36	3479.01	7409102.55	102184.40	3438.24
16	PM-RMMM-PS-LMMM	7556095.14	103983.93	3480.05	7406360.38	102228.86	3437.95
17	PM-PS-LMMM-RMMM	7515881.57	103435.80	3483.89	7399881.73	102140.62	3438.68
18	PM-PS-RMMM-LMMM	7513822.99	103479.53	3483.81	7399236.34	102133.93	3438.36
19	PS-LMMM-RMMM-PM	7559978.72	105089.96	3470.11	7406828.91	102137.04	3438.84
20	PS-LMMM-PM-RMMM	7558180.40	104980.12	3469.33	7406386.82	102161.24	3437.88
21	PS-RMMM-LMMM-PM	7519709.81	103431.85	3482.50	7401020.83	102220.94	3437.68
22	PS-RMMM-PM-LMMM	7516257.85	103405.53	3481.77	7400174.26	102155.29	3438.37
23	PS-PM-LMMM-RMMM	7450970.91	102989.42	3455.84	7396901.61	**102078.33**	3437.08
24	PS-PM-RMMM-LMMM	**7450700.08**	**102986.64**	**3455.20**	**7396602.01**	102079.87	**3437.00**

Table B.3: Neighbourhood structures order testing for 2048x64 dataset using GA-VNS. The best average makespan results are reported in bold.

Case	Nieghbourhood order	2048x64					
		A_u_c_hihi	A_u_i_lohi	A_u_s_lolo	B_u_c_hihi	B_u_i_lohi	B_u_s_lolo
1	LMMM-RMMM-PM-PS	17329758.17	271.58	98.05	5192962.37	27626.06	1023.43
2	LMMM-RMMM-PS-PM	17300487.28	270.21	98.06	5193408.62	27667.20	1023.14
3	LMMM-PM-RMMM-PS	17576466.56	266.50	96.67	5190129.16	27786.74	1020.39
4	LMMM-PM-PS-RMMM	17342620.80	266.55	96.60	5190226.34	27782.87	1021.07
5	LMMM-PS-RMMM-PM	17895612.14	277.24	97.90	5193043.81	27668.28	1023.84
6	LMMM-PS-PM-RMMM	17298123.19	267.94	97.32	5201998.91	27660.77	1017.73
7	RMMM-LMMM-PM-PS	17348268.18	264.77	96.44	5183541.63	27740.10	1014.81
8	RMMM-LMMM-PS-PM	17304399.31	269.30	97.10	5205107.60	27634.76	1023.59
9	RMMM-PM-LMMM-PS	17562688.19	275.93	98.04	5193154.02	27623.50	1023.16
10	RMMM-PM-PS-LMMM	17527866.89	265.16	96.19	5184622.14	27730.96	1015.17
11	RMMM-PS-LMMM-PM	17350083.18	265.83	96.96	5205203.61	27725.04	1018.14
12	RMMM-PS-PM-LMMM	17299174.48	264.70	96.07	5183556.56	27620.64	1014.32
13	PM-LMMM-RMMM-PS	17473300.78	269.38	97.25	5205296.95	27652.03	1018.06
14	PM-LMMM-PS-RMMM	17453691.98	265.93	96.84	5190182.62	27752.64	1020.58
15	PM-RMMM-LMMM-PS	17471395.15	277.24	97.88	5193431.42	27646.29	1024.74
16	PM-RMMM-PS-LMMM	17351215.31	269.27	96.86	5202465.61	27658.80	1017.91
17	PM-PS-LMMM-RMMM	17304924.24	268.15	96.71	5190033.71	27769.42	1020.34
18	PM-PS-RMMM-LMMM	17307863.53	266.50	96.49	5190356.49	27753.35	1019.29
19	PS-LMMM-RMMM-PM	17402234.93	276.24	98.02	5192938.86	27660.52	1024.81
20	PS-LMMM-PM-RMMM	17431016.29	273.99	96.97	5202429.73	27662.95	1017.81
21	PS-RMMM-LMMM-PM	17313463.77	264.47	96.21	5183963.24	26250.05	1014.23
22	PS-RMMM-PM-LMMM	17322004.51	265.96	96.63	5190446.58	27770.68	1018.56
23	PS-PM-LMMM-RMMM	17297647.10	264.58	96.14	5183888.83	27731.93	1015.26
24	PS-PM-RMMM-LMMM	**17296457.38**	**264.40**	**96.06**	**5183526.58**	**26212.04**	**1012.09**

Table B.2: Neighbourhood structures order testing for 1024x32 dataset using GA-VNS. The best average makespan results are reported in bold.

Case	Nieghbourhood order	1024x32					
		A_u_c_hihi	A_u_i_lohi	A_u_s_lolo	B_u_c_hihi	B_u_i_lohi	B_u_s_lolo
1	LMMM-RMMM-PM-PS	19643438.29	509.12	117.67	6006892.32	48406.90	1154.24
2	LMMM-RMMM-PS-PM	19626269.27	509.21	117.82	6007711.65	48302.94	1155.78
3	LMMM-PM-RMMM-PS	19657920.41	523.56	117.76	6007551.36	48330.08	1152.01
4	LMMM-PM-PS-RMMM	19643457.42	509.14	117.53	6006002.32	48433.67	1155.66
5	LMMM-PS-RMMM-PM	19660121.85	513.39	117.70	6003632.80	48326.10	1151.69
6	LMMM-PS-PM-RMMM	19613038.68	510.31	117.71	6003572.70	48314.61	1151.59
7	RMMM-LMMM-PM-PS	19644635.99	510.44	117.56	6011272.23	48469.71	1155.60
8	RMMM-LMMM-PS-PM	19626361.29	519.79	117.85	6008885.71	48749.34	1173.66
9	RMMM-PM-LMMM-PS	19656575.77	524.92	117.66	6017031.42	49110.39	1167.62
10	RMMM-PM-PS-LMMM	19657689.51	539.15	118.13	6015187.21	48339.17	1173.18
11	RMMM-PS-LMMM-PM	19645028.50	510.77	117.97	6010550.78	48461.77	1156.15
12	RMMM-PS-PM-LMMM	19622239.02	515.96	117.52	6008207.79	48290.17	1150.47
13	PM-LMMM-RMMM-PS	19654694.13	540.57	118.09	6047644.20	49422.94	1173.04
14	PM-LMMM-PS-RMMM	19653061.61	519.12	117.95	6006780.22	48498.56	1168.86
15	PM-RMMM-LMMM-PS	19652586.53	535.51	117.77	6016053.12	48935.74	1171.78
16	PM-RMMM-PS-LMMM	19646435.45	518.17	117.69	6008831.75	48402.21	1174.69
17	PM-PS-LMMM-RMMM	19632369.46	511.96	117.74	6007237.32	48407.83	1153.47
18	PM-PS-RMMM-LMMM	19632264.93	519.21	117.87	6028125.68	48630.20	1167.84
19	PS-LMMM-RMMM-PM	19648718.79	538.67	118.14	6013598.95	48565.02	1170.48
20	PS-LMMM-PM-RMMM	19650481.38	535.77	118.02	6009139.61	48838.69	1170.11
21	PS-RMMM-LMMM-PM	19636753.80	520.92	117.73	6008435.59	48388.69	1170.01
22	PS-RMMM-PM-LMMM	19640581.93	513.65	117.55	6005952.55	48441.65	1155.76
23	PS-PM-LMMM-RMMM	19612333.61	522.06	119.74	6014647.15	48598.23	1172.40
24	PS-PM-RMMM-LMMM	**19609730.66**	**507.18**	**117.50**	**6003413.46**	**48288.67**	**1150.38**

Table B.4: Average improvement percentages of ACO-VNS over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	11.91	7.43	3.22	1.03	0.59	-0.06	0.38	0.55
u_c_hilo.0	6.36	1.70	1.12	0.21	0.41	-0.22	0.21	0.52
u_c_lohi.0	12.87	7.12	4.38	2.02	1.54	0.55	1.28	1.52
u_c_lolo.0	6.90	2.06	1.05	0.20	0.29	-0.17	0.35	0.51
u_i_hihi.0	16.34	5.31	7.74	3.88	0.27	0.61	0.57	0.54
u_i_hilo.0	8.97	3.04	3.09	2.12	0.36	0.24	0.45	0.56
u_i_lohi.0	14.99	4.70	7.38	3.17	-0.01	1.36	0.22	0.14
u_i_lolo.0	8.29	2.51	2.87	1.84	0.18	0.13	0.33	0.44
u_s_hihi.0	19.77	9.33	6.43	4.19	0.54	0.69	2.34	1.38
u_s_hilo.0	8.01	2.39	2.15	1.04	0.61	0.01	0.60	0.17
u_s_lohi.0	10.43	3.80	3.36	1.55	-1.39	-1.82	-1.34	-1.37
u_s_lolo.0	10.81	3.74	2.06	0.99	0.17	0.03	0.65	0.36
Avg	11.30	4.43	3.74	1.85	0.30	0.11	0.50	0.44

Table B.5: Average improvement percentages of GA-VNS over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	12.49	8.04	3.86	1.68	1.25	0.60	1.03	1.21
u_c_hilo.0	6.64	2.00	1.42	0.52	0.72	0.09	0.52	0.83
u_c_lohi.0	13.25	7.52	4.80	2.45	1.97	0.99	1.71	1.95
u_c_lolo.0	7.17	2.35	1.33	0.49	0.58	0.12	0.64	0.80
u_i_hihi.0	16.35	5.33	7.76	3.90	0.29	0.63	0.60	0.56
u_i_hilo.0	9.12	3.20	3.25	2.29	0.52	0.41	0.62	0.72
u_i_lohi.0	15.23	4.96	7.64	3.44	0.27	1.64	0.50	0.42
u_i_lolo.0	8.50	2.73	3.09	2.06	0.41	0.36	0.56	0.67
u_s_hihi.0	20.46	10.11	7.23	5.01	1.39	1.54	3.17	2.22
u_s_hilo.0	8.34	2.74	2.50	1.39	0.97	0.37	0.96	0.53
u_s_lohi.0	12.93	6.49	6.05	4.30	1.44	1.02	1.49	1.46
u_s_lolo.0	11.11	4.06	2.39	1.32	0.50	0.36	0.99	0.69
Avg	11.80	4.96	4.28	2.40	0.86	0.68	1.06	1.01

Table B.6: Average improvement percentages and statistical analysis of GA-VNS over ACO-VNS for Braun 512x16 dataset

Instance	Improvement	p-value
u_c_hihi.0	0.66	$<10^{-5}$
u_c_hilo.0	0.31	$<10^{-5}$
u_c_lohi.0	0.44	$<10^{-5}$
u_c_lolo.0	0.29	$<10^{-5}$
u_i_hihi.0	0.02	0.00056
u_i_hilo.0	0.16	$<10^{-5}$
u_i_lohi.0	0.28	$<10^{-5}$
u_i_lolo.0	0.23	$<10^{-5}$
u_s_hihi.0	0.86	$<10^{-5}$
u_s_hilo.0	0.36	$<10^{-5}$
u_s_lohi.0	2.79	$<10^{-5}$
u_s_lolo.0	0.33	$<10^{-5}$
Avg	0.56	

Table B.7: Average improvement percentages of ACO-VNS and GA-VNS over some methods from the literature for the 1024x32 and 2048x64 datasets.

Instance	1024x32						2048x64					
	ACO-VNS			GA-VNS			ACO-VNS			GA-VNS		
	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	minmin	pCHC	TPVNS
A.u_c_hihi	11.95	3.37	2.29	12.76	4.26	3.19	8.45	1.74	-0.56	11.76	5.30	3.08
A.u_c_hilo	9.85	1.20	0.84	13.15	4.82	4.47	7.44	1.50	0.19	10.48	4.74	3.47
A.u_c_lohi	11.57	4.72	3.28	12.76	6.00	4.58	8.46	2.39	0.50	11.10	5.20	3.36
A.u_c_lolo	11.36	8.06	6.16	11.46	8.16	6.27	7.73	0.74	-0.73	11.49	4.78	3.37
A.u_i_hihi	19.83	2.66	2.24	20.13	3.02	2.60	21.58	-0.05	-1.87	24.56	3.75	1.99
A.u_i_hilo	24.34	1.50	1.72	25.08	2.46	2.67	25.28	0.20	0.96	26.23	1.46	2.21
A.u_i_lohi	21.63	0.50	1.72	23.30	2.63	3.82	15.69	-1.13	-1.63	17.89	1.50	1.01
A.u_i_lolo	21.08	2.76	3.14	22.12	4.04	4.41	17.15	-0.98	0.52	19.50	1.89	3.35
A.u_s_hihi	14.97	3.44	2.39	16.96	5.70	4.68	15.46	3.21	0.99	17.15	5.13	2.96
A.u_s_hilo	10.90	3.21	0.86	13.28	5.80	3.51	12.71	2.94	-0.12	15.64	6.20	3.25
A.u_s_lohi	15.03	2.18	1.80	17.72	5.28	4.92	10.16	0.38	-1.35	13.83	4.44	2.79
A.u_s_lolo	13.13	3.44	0.78	16.27	6.94	4.37	15.16	2.57	1.10	16.56	4.17	2.72
B.u_c_hihi	8.99	1.54	1.54	10.43	3.09	3.09	6.82	2.17	0.66	9.91	5.42	3.96
B.u_c_hilo	9.51	1.71	2.04	10.96	3.28	3.60	8.32	1.68	-0.77	11.41	4.99	2.63
B.u_c_lohi	9.46	3.70	2.97	9.62	3.86	3.13	7.89	1.02	0.67	9.14	2.37	2.02
B.u_c_lolo	11.87	4.77	2.56	12.51	5.47	3.27	7.36	1.78	0.28	7.59	2.03	0.53
B.u_i_hihi	24.93	2.70	0.23	26.03	4.13	1.69	17.06	0.54	-0.21	17.69	1.31	0.55
B.u_i_hilo	11.85	2.62	4.18	13.08	3.98	5.52	23.24	0.16	-0.13	23.36	0.32	0.03
B.u_i_lohi	12.67	1.39	1.10	14.74	3.73	3.45	18.88	-1.58	0.94	22.10	2.45	4.88
B.u_i_lolo	15.95	3.10	1.94	17.00	4.31	3.17	25.38	-0.27	0.14	27.81	3.00	3.40
B.u_s_hihi	16.10	2.96	1.87	18.13	5.31	4.25	12.38	1.32	-0.31	16.40	5.84	4.28
B.u_s_hilo	13.89	4.04	5.07	14.71	4.96	5.97	13.13	1.58	-1.12	16.28	5.14	2.54
B.u_s_lohi	14.18	1.43	0.54	19.35	7.35	6.52	14.38	3.56	-0.15	15.78	5.14	1.49
B.u_s_lolo	12.07	4.71	2.01	13.50	6.26	3.61	12.96	0.66	-0.98	17.55	5.90	4.35
Avg	14.46	2.99	2.22	16.04	4.78	4.03	13.88	1.09	-0.12	16.30	3.85	2.68

Table B.8: Average improvement percentages and statistical analysis of GA-VNS over ACO-VNS for the 1024x32 and 2048x64 datasets.

Instance	1024x32		2048x64	
	ACO-VNS	p-value	ACO-VNS	p-value
A.u_c_hihi	0.92	$<10^{-5}$	3.61	$<10^{-5}$
A.u_c_hilo	3.66	$<10^{-5}$	3.28	$<10^{-5}$
A.u_c_lohi	1.34	$<10^{-5}$	2.88	$<10^{-5}$
A.u_c_lolo	0.11	$<10^{-5}$	4.07	$<10^{-5}$
A.u_i_hihi	0.37	$<10^{-5}$	3.79	$<10^{-5}$
A.u_i_hilo	0.97	$<10^{-5}$	1.26	$<10^{-5}$
A.u_i_lohi	2.13	$<10^{-5}$	2.60	$<10^{-5}$
A.u_i_lolo	1.31	$<10^{-5}$	2.84	$<10^{-5}$
A.u_s_hihi	2.34	$<10^{-5}$	1.99	$<10^{-5}$
A.u_s_hilo	2.68	$<10^{-5}$	3.36	$<10^{-5}$
A.u_s_lohi	3.17	$<10^{-5}$	4.08	$<10^{-5}$
A.u_s_lolo	3.62	$<10^{-5}$	1.64	$<10^{-5}$
B.u_c_hihi	1.58	$<10^{-5}$	3.32	$<10^{-5}$
B.u_c_hilo	1.60	$<10^{-5}$	3.37	$<10^{-5}$
B.u_c_lohi	0.17	$<10^{-5}$	1.36	$<10^{-5}$
B.u_c_lolo	0.73	$<10^{-5}$	0.25	$<10^{-5}$
B.u_i_hihi	1.47	$<10^{-5}$	0.77	$<10^{-5}$
B.u_i_hilo	1.40	$<10^{-5}$	0.16	0.00009
B.u_i_lohi	2.37	$<10^{-5}$	3.97	$<10^{-5}$
B.u_i_lolo	1.25	$<10^{-5}$	3.26	$<10^{-5}$
B.u_s_hihi	2.42	$<10^{-5}$	4.58	$<10^{-5}$
B.u_s_hilo	0.96	$<10^{-5}$	3.62	$<10^{-5}$
B.u_s_lohi	6.02	$<10^{-5}$	1.64	$<10^{-5}$
B.u_s_lolo	1.63	$<10^{-5}$	5.27	$<10^{-5}$
Avg	1.84		2.79	

References

[1] I. Foster, C. Kesselman, The history of the grid, computing 20 (21) (2010) 22.

[2] F. Ian, K. Carl, The grid: blueprint for a future computing infrastructure (1999).

[3] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, International journal of high performance computing applications 15 (3) (2001) 200–222.

[4] M. M. Alobaedy, K. R. Ku-Mahamud, Scheduling jobs in computational grid using hybrid acs and ga approach, in: Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE, IEEE, 2014, pp. 223–228.

[5] M. Dorigo, M. Birattari, et al., Swarm intelligence., Scholarpedia 2 (9) (2007) 1462.

[6] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed computing 61 (6) (2001) 810–837.

[8] M. T. Younis, S. Yang, B. Passow, Meta-heuristically seeded genetic algorithm for independent job scheduling in grid computing, in: European Conference on the Applications of Evolutionary Computation, Springer, 2017, pp. 177–189.

[9] O. H. Ibarra, C. E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, Journal of the ACM (JACM) 24 (2) (1977) 280–289.

[10] J. Carretero, F. Xhafa, A. Abraham, Genetic algorithm based schedulers for grid computing systems, International Journal of Innovative Computing, Information and Control 3 (6) (2007) 1–19.

[11] A. Abraham, R. Buyya, B. Nath, Nature?s heuristics for scheduling jobs on computational grids, in:

The 8th IEEE international conference on advanced computing and communications (ADCOM 2000), 2000, pp. 45–52.
[12] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of parallel and distributed computing 59 (2) (1999) 107–131.
[13] F. Xhafa, J. A. Gonzalez, K. P. Dahal, A. Abraham, A ga (ts) hybrid algorithm for scheduling in computational grids., HAIS 9 (2009) 285–292.
[14] F. Xhafa, J. Kolodziej, L. Barolli, A. Fundo, A ga+ ts hybrid algorithm for independent batch scheduling in computational grids, in: Network-Based Information Systems (NBiS), 2011 14th International Conference on, IEEE, 2011, pp. 229–235.
[15] F. Xhafa, J. Carretero, L. Barolli, A. Durresi, Requirements for an event-based simulation package for grid systems, Journal of Interconnection Networks 8 (02) (2007) 163–178.
[16] S. Selvi, D. Manimegalai, Task scheduling using two-phase variable neighborhood search algorithm on heterogeneous computing and grid environments., Arabian Journal for Science & Engineering (Springer Science & Business Media BV) 40 (3).
[17] H. Liu, A. Abraham, A. E. Hassanien, Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, Future Generation Computer Systems 26 (8) (2010) 1336–1343.
[18] S. Selvi, D. Manimegalai, A. Suruliandi, Efficient job scheduling on computational gridwith differential evolution algorithm, International Journal of Computer Theory and Engineering 3 (2) (2011) 277.
[19] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, Task execution time modeling for heterogeneous computing systems, in: Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th, IEEE, 2000, pp. 185–199.
[20] F. Xhafa, A. Abraham, Computational models and heuristic methods for grid scheduling problems, Future generation computer systems 26 (4) (2010) 608–621.
[21] J. Kołodziej, F. Xhafa, Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population, Future Generation Computer Systems 27 (8) (2011) 1035–1046.
[22] S. Nesmachnow, H. Cancela, E. Alba, A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling, Applied Soft Computing 12 (2) (2012) 626–639.
[23] F. Xhafa, J. Kolodziej, L. Barolli, V. Kolici, R. Miho, M. Takizawa, Evaluation of hybridization of ga and ts algorithms for independent batch scheduling in computational grids, in: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on, IEEE, 2011, pp. 148–155.
[24] F. Xhafa, A hybrid evolutionary heuristic for job scheduling on computational grids, in: Hybrid Evolutionary Algorithms, Springer, 2007, pp. 269–311.
[25] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & operations research 24 (11) (1997) 1097–1100.
[26] P. Hansen, N. Mladenović, R. Todosijević, S. Hanafi, Variable neighborhood search: basics and variants, EURO Journal on Computational Optimization 5 (3) (2017) 423–454.
[27] E. Alba, G. Luque, A new local search algorithm for the dna fragment assembly problem, Evolutionary Computation in Combinatorial Optimization (2007) 1–12.
[28] A. B. Ali, G. Luque, E. Alba, K. E. Melkemi, An improved problem aware local search algorithm for the dna fragment assembly problem, Soft Computing 21 (7) (2017) 1709–1720.
[29] S. Nesmachnow, E. Alba, H. Cancela, Scheduling in heterogeneous computing and grid environments using a parallel chc evolutionary algorithm, Computational Intelligence 28 (2) (2012) 131–155.
[30] G. F. Minetti, G. Luque, E. Alba, The problem aware local search algorithm: an efficient technique for permutation-based problems, Soft Computing (2017) 1–14.
[31] J. Eaton, S. Yang, Dynamic railway junction rescheduling using population based ant colony optimisation, in: Computational Intelligence (UKCI), 2014 14th UK Workshop on, IEEE, 2014, pp. 1–8.
[32] M. Dorigo, T. Stützle, The ant colony optimization metaheuristic: Algorithms, applications, and advances, in: Handbook of metaheuristics, Springer, 2003, pp. 250–285.
[33] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, I. de la Iglesia, A. Perallos, Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial

optimization problems, The Scientific World Journal 2014.
[34] M. T. Younis, S. Yang, A genetic algorithm for independent job scheduling in grid computing.
[35] F. Xhafa, E. Alba, B. Dorronsoro, B. Duran, A. Abraham, Efficient batch job scheduling in grids using cellular memetic algorithms, in: Metaheuristics for Scheduling in Distributed Computing Environments, Springer, 2008, pp. 273–299.
[36] F. Xhafa, J. Carretero, B. Dorronsoro, E. Alba, A tabu search algorithm for scheduling independent jobs in computational grids, Computing and informatics 28 (2) (2012) 237–250.