A Loosely Coupled Hybrid Meta-Heuristic Algorithm for the Static Independent Task Scheduling Problem in Grid Computing

Muhanad Tahrir Younis*, Shengxiang Yang†, and Benjamin N. Passow‡
Centre for Computational Intelligence, School of Computer Science and Informatics
De Montfort University, The Gateway, Leicester LE1 9BH, United Kingdom
Email: *muhanad.younis@dmu.ac.uk, †syang@dmu.ac.uk, ‡benpassow@ieee.org

***Abstract*—Task scheduling is one of the most difficult problems in grid computing systems. Therefore, various studies have been proposed to present methods which provide efficient schedules. Meta-heuristic approaches are among the methods which have proven their efficiency in this domain. However, the literature shows that hybridizing two or more meta-heuristics can improve performance to a greater extent than stand-alone algorithms as the new high-level algorithm will inherit the best features of the hybridized algorithms. In this paper, a loosely coupled hybrid meta-heuristic algorithm is proposed for solving the static independent task scheduling problem in grid computing. It combines ant colony optimization and variable neighborhood search, where the former operates first and whose output is subsequently improved by the latter. The experimental results show that the proposed algorithm achieves better task-machine mapping in terms of minimizing makespan than other selected approaches from the literature.**

***Index Terms*—Hybrid meta-heuristic, Ant Colony Optimization, Variable Neighborhood Search, Task Scheduling**

I. Introduction

Grid Computing provides a type of parallelized distributed infrastructure which allows the geographically distributed autonomous and heterogeneous machines to be shared, selected and aggregated dynamically depending on their availability, capability, performance, cost, and users' quality-of-service requirements to create a virtual supercomputer which is able to efficiently solve various complex problems from commercial and non-commercial clients. Grid computing was mainly developed to fulfill the significant increase in requirements for high computing power from various private organizations and the scientific computing community [1].

Grid computing has witnessed several developments since the introduction of its early definitions in [2], [3]. The developments are aimed at a better understanding of the grid issues through the enhancement of the grid infrastructure and middleware. Allocating tasks, also called jobs or applications, to computational grid machines in an efficient manner is one of the main challenges facing any computational grid system; this allocation is called task scheduling in grid computing. An efficient scheduler is one which can make practical and effective use of the available distributed resources. These resources are connected through heterogeneous environments in an efficient, reliable and secure manner. Similar to task scheduling in traditional computing systems, this allocation is known to be an NP complete problem [4]; however, it is made more complicated in grid computing due to its dynamic nature, high degree of task and machine heterogeneity, problem size, and other factors such as existing local schedulers and policies [5].

Several stand alone meta-heuristics, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Variable Neighborhood Search (VNS) and Tabu Search (TS), have been applied successfully for various types of scheduling problems. However, the results achieved by these methods could be further improved by combining two or more meta-heuristics [6]. The resulting new high-level algorithm would then inherit the best features of the combined meta-heuristics. Consequently, the chances of escaping from a local minimum will be increased, and hence the overall performance will be enhanced [7].

In general, there are two means, or fashions by which to effectively combine meta-heuristics, namely loosely coupled and strongly coupled [8]. The first fashion consists of executing the combined meta-heuristics in a serial manner such that the solution to the first method then being used by the second and so on; the final solution will be the output of the last algorithm. The second fashion refers to the type of hybridization in which the inner procedures of the hybridized algorithms are interchanged in such a way that one of the methods acts as the main algorithm, which during its execution calls other methods to act as supporting algorithms [9].

In our previous work [10], a loosely coupled hybrid algorithm which combined a newly proposed ACO and the GA proposed in [11], i.e., ACO+GA, was suggested as a promising algorithm for solving the task scheduling problem in grid computing. The ACO starts first, the output from which will be used by the GA proposed in [11], which further improves it. The Expected Time to Compute (ETC) simulation model was used to evaluate the proposed method in terms of minimizing the makespan by generating our own 512x16 dataset instances using the range-based method described in [12].

In this work, the use of VNS for task scheduling in grid computing is introduced. Four new neighborhood structures, together with a modified local search, are proposed. The proposed VNS is hybridized with a meta-heuristic method in a loosely coupled fashion, yielding a new hybrid meta-heuristic algorithm by which to consider the task scheduling problem

in grid computing. The new algorithm, called ACO+VNS, combines a modified ACO from our previous work [10] and VNS, in which the former works first and whose output is further refined by the latter algorithm. To evaluate the proposed method in terms of minimizing the makespan, the ETC simulation model is used to carry out a number of experiments. A well known dataset will be used for this purpose rather than generating a special dataset so that we can easily make a fair comparison to some current state-of-the-art methods.

The remainder of this paper is organized as follows. Section II introduces the related work on solving the task scheduling problem in grid computing using meta-heuristic methods. The simulation model used to evaluate the performance of the proposed method is explained in Section III. Section IV presents the use of VNS for task scheduling in grid computing, while Section V similarly describes the use of ACO. Parameter setting is discussed in section VI. Section VII presents the experimental results obtained by applying the proposed method. Finally, a set of conclusions are provided in Section VIII.

II. Related work

There are a number of different approaches described in the literature which have tackled the task scheduling problem on computational grid systems such as simple queuing algorithms and heuristic algorithms. However, the application of meta-heuristic techniques to the task scheduling problem in these systems is preferred because they can efficiently cope with the complexity inherent to this problem [13]. Meta-heuristic algorithms are high-level search methods which are applicable to solve a wide range of NP complete problems. In a practical sense, these high-level methods are considered to be most likely to be the best candidate to deal with the complexity of task scheduling in grid computing, and hence many approaches have been suggested, such as VNS, ACO and GA, to name just a few [14].

The authors in [15] suggested a Multi-objective VNS (MVNS) method to solve the task scheduling problem on grid. They have considered two objectives, namely makespan and flowtime, to minimize. Several neighborhood structures have been introduced and an effective local search, called the random Problem Aware Local Search Heuristic (PALS), was employed. The performance of the proposed method has been compared with some methods described in the literature, the results of which show that MVNS outperforms all the compared methods in all of the cases tested.

A Two-Phase VNS (TPVNS) algorithm for task scheduling on heterogeneous computing and grid systems was presented in [16]. The authors also used the random PALS together with six neighborhood structures. Some approaches from the literature have been selected by the authors for comparison with TPVNS in terms of performance. The results show that TPVNS outperforms them all in the majority of cases investigated.

A hybrid method, called ACO+TS, which combines ACO and TS in a loosely coupled fashion, was proposed in [17]. The experimental results show that the use of TS with ACO improves the quality of the solutions; however, the hybrid method took over 3.5 hours to achieve these results, while the authors in [6] suggested a loosely coupled hybrid meta-heuristic, called ACO+GA, for the task scheduling on computation grid, which combines ACO and GA. However, a non-standard dataset was considered and the code used for their implementation is not available to allow for an even-handed comparison.

The work presented in [11] studied the application of several static heuristics to the task scheduling problem in heterogeneous environments with the aim of minimizing the makespan. Their experiments show that the proposed GA achieves better results than the other heuristics used in the work. A population of 200 solutions was used, which was generated either randomly or by seeding with one solution generated by a min-min algorithm [18], whilst the reminding 199 solutions were generated randomly.

The study presented in [5] examined the use of GA for minimizing the multi-objective task scheduling problem in grid systems. In order to provide diversity, the authors used two deterministic heuristics, which are the Longest Job to Fastest Resource−Shortest Job to Fastest Resource (LJFR−SJFR) [19] and the Minimum Completion Time (MCT) [20], and also the random method to generate the initial population. Moreover, a variety of GA operators and encoding schemes were examined by the authors.

Two hybrid meta-heuristics were proposed in [8] and [9] to address the task scheduling problem in computational grids. The former method combines GA and TS in a strongly coupled fashion in which GA is the main algorithm which calls TS during its execution to further enhance the quality of the solutions in the population. On the other hand, the latter method combines the same methods in a loosely coupled fashion, that is, GA is executed first and its final solution is further improved by TS. The performance of both proposals was evaluated using the HyperSim-G grid simulator [21].

III. Simulation model

In this paper, the ETC model is used which was introduced in [12] and has been used to study the problem of static task scheduling algorithms for heterogeneous computing, such as grid computing, in [11]. In this model, it is assumed that an accurate estimation or prediction of the size of each task, the computing power of each machine, and an estimation of the load on the machines are known in advance. Furthermore, an accurate estimation of the expected execution time for each task on each machine should either be computable or is assumed to be known beforehand. These assumptions are realistic since it is relatively straightforward to gather information about the tasks' requirements and the computational power of machines from the specifications provided by the user, by predications, or from historic data [9]. This estimation is represented in a two-dimensional array called ETC, where

$ETC[t][m]$ indicates the expected execution time that task t needs to finish on machine m.

A description of the problem under the ETC model can be formulated as follows:

1) A set of n independent tasks $T = \{t_0, t_1, ..., t_{n-1}\}$ to be assigned to grid machines. Any task can be handled by any machine. However, these tasks are non-preemptive, i.e., each task should be executed entirely by one machine only.
2) A set of r heterogeneous machines $M = \{m_0, m_1, ..., m_{r-1}\}$ to be used for processing the n independent tasks.
3) The ETC matrix has a size $n \times r$, where $ETC[t][m]$ denotes the estimated required time for processing task t by machine m.
4) The goal of task scheduling in grid computing is to find a mapping of the submitted tasks onto the available machines that minimizes the makespan, which represents the finishing time of the latest task and can be computed as follows:

$$makespan = min_{s \in S} max_{t \in T}(Finish\_t), \quad (1)$$

where S is the set of all possible solutions and $Finish\_t$ represents the time by which task t will be completed [13].

The ETC matrix can be generated in a straightforward manner by dividing the size of a given task t by the computing power of a machine m. One example of this type is the dataset presented in [22]. However, the authors in [12] proposed a method to generate ETC matrices called the range-based method. Through the use of three different types of metrics, namely task heterogeneity, machine heterogeneity and consistency, the method captures the various characteristics of grid computing systems. Therefore, twelve distinct ETC matrices are needed so that we can consider all these various characteristics.

An example of range-based ETC matrices is the 12 classic instances proposed in [11]. Each instance has 512 tasks and 16 machines. The following abbreviation has been used to identify the type of ECT matrix, D-T-THMH.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.
- TH denotes the heterogeneity of the tasks, with two possibilities, either hi for high or lo for low.
- MH denotes the heterogeneity of the machines, with two possibilities, either hi for high or lo for low.

IV. Applying VNS to the task scheduling problem

VNS is a simple and effective meta-heuristic algorithm that is often applied to many optimization problems; Mladenovic and Hansen [23] proposed VNS in 1997 as a flexible framework which can be used to define heuristics that are applicable to various problems. VNS uses multiple neighborhood structures to explore a number of neighborhoods for the current incumbent solution. It then picks the neighbor that introduces an improvement. The systematic change of neighborhood structures is the core concept of the VNS meta-heuristic. This change takes place in the descent phase where the meta-heuristic seeks to find a local minimum; it also occurs in the perturbation phase where the VNS attempts to escape from the local minimum.

Algorithm 1 The VNS algorithm

```
1: Let: S be the initial solution and N_k be the set of
   neighborhood structures, k = 1, ..., k_max
2: repeat
3:     k ← 1;
4:     repeat
5:         Ś ← shake(S);
6:         S̃ ← local_search(Ś);
7:         if (fitness(S̃) < fitness(S)) then
8:             S ← S̃;
9:             k ← 1;
10:        else
11:            k ← k+1;
12:        end if
13:    until (k==k_max)
14: until (termination condition)
```

The pseudocode of the VNS meta-heuristic is illustrated in Algorithm 1. Generally, the VNS involves three steps, which are repeated until the termination conditions are satisfied. The first step, known as the shake step, includes the application of a set of operators in a particular order. Applying these operators allows the VNS meta-heuristic to resolve from the local minimum traps. The second step, called the improvement step, involves performing a local search that attempts to enhance the solution found in the shaking step. The two most common improvement procedures are the first improvement and the best improvement. In the first improvement, the local search procedure ends when an improvement to the current solution is found. On the other hand, the best improvement examines all possible solutions and chooses the best among them. Finally, the third step in the VNS meta-heuristic is the neighborhood change step, which is used to make a decision about the neighborhood that will be explored next and whether to accept the current solution as a new incumbent solution or not. Several neighborhood change procedures are available in the literature, such as the sequential, cyclic and pipe neighborhood change procedures [24].

A. Neighborhood structures for task scheduling in grid computing

The neighborhood structure provides a means by which to explore new regions of 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 neighborhoods; thus, the use of well-defined neighborhood structures will certainly lead to better exploration.

In the task scheduling problem, for every solution X there is at least one machine with a local makespan time equals to the overall makespan of the solution, which is known as the 'problem machine'. New solutions can be obtained from X by exchanging a task currently allocated to the problem machine with another task allocated to another machine, or by moving a task currently allocated to the problem machine to any other machine. Therefore, we can define various new neighborhood structures based on the concepts of exchange and move. In this work, we proposed four neighborhood structures, namely the Best Exchange (BE), the Maximum Heavy to Light Move (MHLM), the Random Heavy to Light Move (RHLM) and the Best Move (BM).

BE alters the solution by finding the set of exchanges for some of the tasks allocated to the problem machine with those allocated to other machines to best improve the solution in terms of minimizing the makespan. The second structure, MHLM, defines new neighbors by moving the task in the list of tasks assigned to the problem machine which has the maximum expected completion time to the machine which has the minimum processing time for the same. On the other hand, RHLM, the third neighborhood structure, moves a random task from the list of the tasks assigned to the problem machine to the machine with the minimum local makespan time. Finally, BM modifies the solution by finding the set of moves which allocate some of the tasks assigned to the problem machine to other machines such as to best reduce the makespan time.

B. Improvement procedure

This procedure includes the application of a local search to enhance the solution found by the shaking step. For this purpose, we used a modified version of the Problem Aware Local Search (PALS) algorithm. PALS was originally proposed to solve the problem of DNA fragment assembly [25], [26]. In [27] and [16], a variant of PALS, called Randomized PALS, has been used for task scheduling on heterogeneous environments. Recently, it has been used as an efficient technique for some permutation-based optimization problems [28].

From a given solution S, the algorithm picks two machines, Pm and Rm, where Pm is the machine with the largest local makespan and Rm is a random machine such that $Pm \neq Rm$. The outer loop iterates on some of the tasks of Pm. The number of these tasks is determined by generating two random numbers, Pm_start and Pm_end, such that $Pm\_start \in$ [1, $Pm\_size - 1$] and $Pm\_end \in [Pm\_start, Pm\_size]$, where Pm_size is the number of the tasks allocated to Pm. Similarly, the inner loop works on the tasks assigned to Rm using Rm_start and Rm_end, which are generated in the same manner as Pm_start and Pm_end, respectively. This process, which was suggested in [16], guarantees the selection of different tasks with different sizes in each iteration, while the randomize PALS used in [27] has used a different method whereby the start task of each loop is generated randomly and the number of tasks involved in the outer loop is fixed to 32 and to $NT/20$ in the inner loop, where NT is the total number of tasks. The double loop calculates the makespan values when swapping machines and transferring tasks in S1 and S2, respectively. Then it selects the solution with the minimum makespan and compares this with the best solution found to that point; if there is an improvement, then it will become the new best; otherwise, the process will continue. Therefore, this loop stores the best improvement to the solution with respect to the makespan time obtained by applying (Pm_end x Rm_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 times, which means that the best improvement is applied, whereas the random PALS used in [27] and [16] is applied until an improvement on the original solution is discovered or until max iterations have been performed, i.e., the first improvement strategy was used. Algorithm 2 lists the pseudocode of the modified random PALS.

Algorithm 2 The PALS algorithm

```
for (iter = 1 to max) do
    sol_ms ← makespane(S);
    S_0 ← S;
    min_ms ← ∞;
    Find the problem machine (Pm) which has the maximum local makespan time;
    Pm_List ← the tasks assigned to Pm;
    Pm_size ← Pm_List size;
    Randomly select a machine Rm such that Rm ≠ Pm;
    Rm_List ← the tasks list of Rm;
    Rm_size ← Rm_List size;
    Pm_start ← rand(1, Pm_size − 1);
    Pm_end ← rand(Pm_start, Pm_size);
    Rm_start ← rand(1, Rm_size − 1);
    Rm_end ← rand(Rm_start, Rm_size);
    for (i=Pm_start to Pm_end) do
        for (j=Rm_start to Rm_end) do
            S_1 ← swap_machines(S_0,Pm_List[i],
                                Rm_List[j]);
            S_2 ← transfer_task(S_0,Pm_List[i], Rm);
            if (makespan(S_1) < makespan(S_2)) then
                S_3 ← S_1;
            else
                S_3 ← S_2;
            end if
            curr_ms ← makespan(S_3);
            if (curr_ms < min_ms) then
                S_0 ← S_3;
                min_ms ← curr_ms.
            end if
        end for
    end for
    if (min_ms < sol_ms) then
        S ← S_0.
    end if
end for
```

V. Proposed ACO+VNS for task scheduling

ACO is a meta-heuristic search algorithm which simulates the behavior of ants in the process of foraging for food [29]. ACO seems an appropriate candidate for the problem of task scheduling in computational grids as it has been successfully used for various NP complete optimization problems. This section introduces a loosely coupled hybrid ACO+VNS algorithm for the problem of task scheduling in grid systems, which uses the modified ACO described in our previous work [10].

The ACO algorithm uses the pheromone trail and the heuristic function information to construct a solution to an optimization problem. Determining what information the pheromone trail encodes is the first step in any ACO-based algorithm. The pheromone trail is used by ants to share useful information about good solutions. A pheromone matrix, τ, is required to hold n tasks and r machines, where $\tau[t][m]$ indicates the favorability of assigning task t to machine m. The second set of information the ants use to build their solutions is the heuristic function, η_{tm}. The following heuristic function has been used for the task scheduling problem:

$$\eta_{tm} = \frac{1}{free[m]}, \tag{2}$$

where the list $free[m]$ represents the time by which the machine m will become free. η_{tm} will be large if $free[m]$ is small. Thus, a machine will be more desirable if it is freed earlier.

The ACO algorithm also uses another function, the fitness function, to measure the quality of the solutions. In this work, minimizing the makespan is considered, which represents the general throughput of a grid computing system. A small value for the makespan indicates that the scheduler is producing a high-quality schedule that efficiently maps tasks to machines.

The pheromone trail is updated after each iteration according to Equation (3). This update allows ants to share information about the current states of the machines.

$$\tau_{tm} = \begin{cases} \rho * \tau_{tm} + \Delta\tau_{tm} & \text{if m} \leftarrow \text{t in } lbest\_ant \\ \rho * \tau_{tm} & \text{otherwise,} \end{cases} \tag{3}$$

where ρ ($0 < \rho \leq 1$) represents the decay parameter the ants use to forget poor information and $\Delta\tau_{tm}$ denotes the amount of pheromone deposit on the path, which can be computed as follows:

$$\Delta\tau_{tm} = \frac{makespan(lbest\_ant)}{makespan(gbest\_ant)} \tag{4}$$

It is worth noting that we used different rules for updating the pheromone trail and for $\Delta\tau_{tm}$ in our previous work [10].

Two lists are used by each ant to build a solution. The first list is called the scheduled list, which is initially empty, whilst the second list is the unscheduled list, and initially contains n tasks. The first task-machine pair is randomly selected. The pheromone trail τ_{ab} provides each ant with information about the favorability of assigning task x to machine y. On the other hand, the heuristic function, η_{ab}, will find the best available machine b, in terms of being free earlier, to process a task a from the unscheduled tasks list. Furthermore, the inverse of $ETC[a, b]$ is used as an additional heuristic function; the inverse was used since lower values are more preferable. A task a is probabilistically selected to be allocated to a machine b using the transition rule defined as follows:

$$p_{ab} = \frac{[\tau_{ab}]^{\alpha} * [\eta_{ab}]^{\beta} * \frac{1}{ETC[a,b]}}{\sum[\tau_{ab}]^{\alpha} * [\eta_{ab}]^{\beta} * \frac{1}{ETC[a,b]}}, \tag{5}$$

where α and β are the relative weight parameters of the pheromone and heuristic information, respectively.

This process continues until the unscheduled tasks list becomes empty. The same procedure is followed by each ant in the colony. When all ants have built their solutions, the best local ant in the colony is determined; that is, the one with the minimum makespan. The pheromone trail update rule is then applied as described in Eq. (3). The above steps are repeated until some stopping conditions become true. The final solution found by ACO will be further improved by applying the VNS algorithm to it. Algorithm 3 demonstrates the general steps of

Algorithm 3 The proposed ACO+VNS algorithm for task scheduling in grid computing

```
Let n and r represent the number of tasks and machines respectively;
Initialize the pheromone trail τ_tm to a small value;
Initialize free[0..r − 1] to 0;
Initialize the decay parameter ρ;
gbest_ant ← min_min();
makespan ← the makespan of gbest_ant;
Δτ_tm ← 1/makespan;
Update the pheromone trail using Eq. (3);
while (the stopping condition is not true) do
    for (each ant) do
        Randomly select the task-machine pair (u, v) and add it to the scheduled list;
        for (all unscheduled tasks) do
            free[v] ← free[v] + ETC[u, v];
            Compute η_vu using Eq. (2);
            Compute ρ_ab using Eq. (5);
            Find the largest value of ρ_ab;
            Select the next task-machine pair (u = a, v = b) and add it to the scheduled list;
        end for
    end for
    Compute the makespan values for each ant using Eq. (1);
    Find the local best ant lbest_ant.
    if ((makespan(lbest_ant) < makespan(gbest_ant)) then
        gbest_ant ← lbest_ant
    end if
    Calculate Δτ_tm using Eq. (4);
    Update the pheromone trail using Eq. (3);
end while
Perform VNS: gbest_ant ← VNS(gbest_ant).
```

the proposed hybrid ACO+VNS algorithm. It is worth noting here that in our previous work [10], a small value was used to initialize the pheromone deposit only. In this study, Eq. (3) is also applied before starting the main ACO+VNS procedure, where the solution found by the deterministic heuristic min-min algorithm [18] is used to update the pheromone trail in order to speed up the process of finding good solutions.

VI. Parameters setting

A fixed set of parameters was selected for the proposed hybrid meta-heuristic to set the parameters. This set was tested by running a number of experiments using a number of instances with diverse characteristics. One of the main advantages of VNS is that it does not need many parameters. The stopping condition, which is the maximum number of iterations, was set to 200. Since we used the forward VNS version in this paper, it is necessary to test the order of neighborhood structures. The forward VNS means that the algorithm begins with $k = 1$, which is then incremented by one if no improvement is found; otherwise, we re-set $k = 1$. As we use four neighborhood structures, we have 24 possible combinations. The primarily experiments show that the best results were reported when the following order is used: BE-BM-RHLM-MHLM. For the ACO algorithm, the population size, α, β, and ρ were among the tested parameters. The population size is set to 2 due to our attempt to reduce the computational time needed to construct solutions by ants and increase the number of generations. The authors in [17] tested values of α and β in the range [1, 50]. The best results were achieved when $\alpha = 10$ and $\beta = 10$. In this paper, we selected three values for α and β, namely 1, 5 and 10, for testing. The results show that $\alpha = 10$ and $\beta = 1$ represent the best combination. Two values were used for ρ, which were 0.5 and 0.7. The best makespan values were achieved when using $\rho = 0.7$.

VII. Experimental Results

This section discusses the experimental results of applying the proposed hybrid meta-heuristic method to address the task scheduling problem in a computational grid. The Java language was used to implement the proposed method and the experiments provided in this study were performed using an Intel i5-4570 CPU @3.20GHz with 8 Gigabyte RAM.

In addition to the best, average, and standard deviation, two measures will also be used to compare the results obtained by applying the proposed hybrid method and various other methods from the literature. The first measure is the improvement percentage of one algorithm over another, which can be computed as follows:

$$Improvement(\%) = \frac{algorithm1 - algorithm2}{algorithm1} * 100\%, \quad (6)$$

where algorithm1 and algorithm2 are the makespan values of two different algorithms. The second measure is the relative gap value of any algorithm with respect to the corresponding lower bound, which can be calculated as follows:

$$gap = \frac{Res - LowerBound}{LowerBound}, \quad (7)$$

where Res indicates the makespan value (best or average) obtained by the proposed algorithm for the corresponding test instance and the lower bound reported in the literature.

To test the performance of the proposed method, the dataset from [11], which involves the classical 12 problem instances, has been used. Each instance has 512 tasks and 16 machines. ACO+VNS was compared to a number of algorithms selected from the literature: the min-min algorithm [18], GA [11], Cellular Memetic Algorithms (cMA) [30], Memetic Algorithm and TS (MA+TS) [31], ACO+TS [17], parallel Cross-generational Heterogeneous recombination Cataclysmic mutation (pCHC) [27], and Two-Phase VNS (TPVNS) [16]. The authors in [11] used a GA with a population of 200, and the algorithm stops when it has executed 1000 iterations (number of evolutions) or the best solution has not changed in 150 iterations. The authors in [17] allowed ACO+TS to run for 1000 iterations followed by 1000 iterations of TS. The authors in [16] set 150 seconds as the TPVNS runtime in order to provide results in which about 1000 iterations were performed. cMA [30], MA+TS [31] and pCHC [27] used a fixed 90 seconds to report their results, though there is no information available as to the total number of iterations each algorithm performed in this time. In this paper, ACO was allowed to run for 1000 iterations followed by 200 iterations of VNS, which took about 8 minutes. To obtain the best, average and standard deviation values, ACO+VNS was executed 50 times for each problem instance. Table I provides the results of applying ACO+VNS compared to the other selected methods. The best results are indicated in bold, from which it is clear that the average makespan values achieved by ACO+VNS outperform the other selected methods in 11 out of 12 instances. The ACO+VNS algorithm is expected to find high-quality schedules in any single execution since it has very small standard deviation values, which lie in the range [0.06, 1.01].

Table II shows the improvement percentages of ACO+VNS over the selected methods from the literature. ACO+VNS has a better improvement percentage than all the other methods compared, with a minimum average improvement of 0.19.

In Table I, the last column represents the lower bound values of each problem instance which were reported in [27]. Table II summarizes the gap values between the average makespan results for the proposed methods and some selected algorithms from the literature and their corresponding lower bounds. ACO+VNS achieved the smallest average gap values to the lower bound with a value of 1.54, which indicates that the quality of the solutions it finds are very high compared to the others.

VIII. Conclusions and future work

Task scheduling in grid computing is a complex problem, which can become more complicated due to the fact that it

TABLE I
MAKESPAN RESULTS FOR BRAUN ET AL. [11] DATASET INSTANCES.

Instance	Min-min	GA	cMA	MA+TS	ACO+TS	pCHC		TPVNS		ACO+VNS			Lower Bound
		avg	avg	avg	best	best	avg	best	avg	best	avg	std	
u_c_hihi.0	8460675.00	8050844.50	7700929.80	7530020.20	7497200.90	7461819.10	7481194.50	7439471.80	7494257.80	**7431111.34**	**7453099.03**	0.21	7346524.20
u_c_hilo.0	164022.44	156249.20	155334.80	153917.20	154234.60	153791.90	153924.00	153270.10	154400.30	**153249.98**	**153713.01**	0.23	152700.40
u_c_lohi.0	275837.34	258756.80	251360.20	245288.90	244097.30	241524.00	243446.30	240803.30	244043.20	**240500.86**	**242991.77**	0.22	238138.10
u_c_lolo.0	5546.26	5272.30	5218.20	5173.70	5178.40	5177.50	5181.60	5154.80	5190.30	**5153.00**	**5173.01**	0.22	5132.80
u_i_hihi.0	3513919.25	3104762.50	3186664.70	3058474.90	2947754.10	2952493.20	2956905.70	2944074.60	2955764.70	**2942987.16**	**2955283.00**	0.23	2909326.60
u_i_hilo.0	80755.68	75816.10	75856.60	75108.50	73776.20	73639.80	73847.10	73378.00	73927.00	**73374.91**	**73625.74**	0.18	73057.90
u_i_lohi.0	120517.71	107500.70	110620.80	105808.60	102445.80	102136.10	102677.30	102057.50	102599.70	**102055.23**	**102554.42**	0.22	101063.40
u_i_lolo.0	2779.09	2614.40	2624.20	2596.60	2553.50	2549.80	2557.20	2547.90	2560.10	**2543.60**	**2552.02**	0.16	2529.00
u_s_hihi.0	5160343.00	4566206.00	4424540.90	4321015.40	4162547.90	4198779.50	4239146.30	4145941.70	4197996.50	**4144999.88**	**4167862.59**	0.30	4063563.70
u_s_hilo.0	104540.73	98519.40	98283.70	97177.30	96762.00	96623.30	96750.30	**95872.30**	96330.40	96118.69	**96178.73**	0.06	95419.00
u_s_lohi.0	140284.48	130616.50	130014.50	127633.00	123922.00	123251.50	123989.40	**122986.00**	**123954.30**	123595.16	125729.82	1.01	120452.30
u_s_lolo.0	3867.49	3583.40	3522.10	3484.10	3455.20	3450.10	3472.20	3440.50	3461.90	**3440.02**	**3450.20**	0.22	3414.80

TABLE II
AVERAGE IMPROVEMENT PERCENTAGES OF ACO+VNS OVER VARIOUS METHODS SELECTED FROM THE LITERATURE.

Instance	Min-min	GA	cMA	MA+TS	ACO+TS	pCHC	TPVNS
u_c_hihi.0	11.91	7.42	3.22	1.02	0.38	0.38	0.55
u_c_hilo.0	6.29	1.62	1.04	0.13	0.64	0.14	0.45
u_c_lohi.0	11.91	6.09	3.33	0.94	1.47	0.19	0.43
u_c_lolo.0	6.73	1.88	0.87	0.01	0.49	0.17	0.33
u_i_hihi.0	15.90	4.81	7.26	3.37	0.16	0.05	0.02
u_i_hilo.0	8.83	2.89	2.94	1.97	0.54	0.30	0.41
u_i_lohi.0	14.91	4.60	7.29	3.08	0.38	0.12	0.04
u_i_lolo.0	8.17	2.39	2.75	1.72	0.29	0.20	0.32
u_s_hihi.0	19.23	8.72	5.80	3.54	0.42	1.68	0.72
u_s_hilo.0	8.00	2.38	2.14	1.03	0.66	0.59	0.16
u_s_lohi.0	10.38	3.74	3.30	1.49	0.26	-1.40	-1.43
u_s_lolo.0	10.79	3.72	2.04	0.97	0.44	0.63	0.34
Avg	11.09	4.19	3.50	1.61	0.55	0.25	0.19

TABLE III
THE GAP VALUES TO THE LOWER BOUND OF THE AVERAGE MAKESPAN ACHIEVED BY ACO+VNS AND VARIOUS ALGORITHMS SELECTED FROM THE LITERATURE.

Instance	Min-min	GA	cMA	MA+TS	ACO+TS	pCHC	TPVNS	ACO+VNS
u_c_hihi.0	15.17	9.59	4.82	2.50	2.05	1.83	2.01	1.45
u_c_hilo.0	7.41	2.32	1.73	0.80	1.00	0.80	1.11	0.66
u_c_lohi.0	15.83	8.66	5.55	3.00	2.50	2.23	2.48	2.04
u_c_lolo.0	8.06	2.72	1.66	0.80	0.89	0.95	1.12	0.78
u_i_hihi.0	20.78	6.72	9.53	5.13	1.32	1.64	1.60	1.58
u_i_hilo.0	10.54	3.78	3.83	2.81	0.98	1.08	1.19	0.78
u_i_lohi.0	19.25	6.37	9.46	4.70	1.37	1.60	1.52	1.48
u_i_lolo.0	9.89	3.38	3.76	2.67	0.97	1.12	1.23	0.91
u_s_hihi.0	26.99	12.37	8.88	6.34	2.44	4.32	3.31	2.57
u_s_hilo.0	9.56	3.25	3.00	1.84	1.41	1.40	0.96	0.80
u_s_lohi.0	16.46	8.44	7.94	5.96	2.88	2.94	2.91	4.38
u_s_lolo.0	13.26	4.94	3.14	2.03	1.18	1.68	1.38	1.04
Avg	14.43	6.04	5.28	3.21	1.58	1.80	1.73	1.54

can be a dynamic, multi-objective and contain a high degree of heterogeneity in terms of tasks and machines. Hence, it is necessary to use meta-heuristic approaches in order to cope in practice with its difficulty and complexity. Several stand-alone meta-heuristics 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 VNS, have been hybridized in a loosely coupled fashion to solve the static independent task scheduling problem on computational grid systems. The hybridization of two meta-heuristics in a loosely coupled fashion has the property that each of the combined methods preserves its identity; however, the resultant high-level hybrid algorithm inherits the best characteristics of the combined methods. The concepts of move and exchange for some tasks to or from the problem machine were used for the design of the neighborhood structures and the local search of the VNS. The use of VNS has improved the performance of the ACO algorithm by allowing it to search new parts of the problem's state space.

The ETC model is used to analyze the performance of the proposed method in terms of minimizing the makespan. The experiments show that ACO+VNS obtained results which are better than the other approaches selected from the literature. The low standard deviations found for the reported results suggest that ACO+VNS can achieve high-quality schedules in any given run. Furthermore, ACO+VNS obtained results that have the smallest gap to the lower bound compared to the other selected methods in all problem instances examined in this study; however, it needs a longer time to construct these solutions.

Although the proposed method represents a promising ap-

proach to solving the task scheduling problem in grid environments, the work in this study could be extended along many dimensions. One such dimension includes examining the proposed method in a strongly coupled fashion in order to compare which hybridization scheme is better. Moreover, adding another objective, so that the problem will be multi-objective, and testing it in a dynamic environment may also be considered in future work. Furthermore, the proposed method presented in this study was implemented sequentially. One possible extension would be testing the performance of the proposed method in parallel mode.

REFERENCES

[1] I. Foster and C. Kesselman, "The history of the grid," *Computing*, vol. 20, no. 21, p. 22, 2010.

[2] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[4] F. Xhafa and A. Abraham, "Computational models and heuristic methods for grid scheduling problems," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 608–621, 2010.

[5] J. Carretero, F. Xhafa, and A. Abraham, "Genetic algorithm based schedulers for grid computing systems," *International Journal of Innovative Computing, Information and Control*, vol. 3, no. 6, pp. 1–19, 2007.

[6] M. M. Alobaedy and 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.

[7] F. Xhafa, J. Kolodziej, L. Barolli, V. Kolici, R. Miho, and 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.

[8] F. Xhafa, J. A. Gonzalez, K. P. Dahal, and A. Abraham, "A GA (TS) hybrid algorithm for scheduling in computational grids." *HAIS*, vol. 9, pp. 285–292, 2009.

[9] F. Xhafa, J. Kolodziej, L. Barolli, and A. Fundo, "A ga+ ts hybrid algorithm for independent batch scheduling in computational grids," in *Proceedings 14th Int. Conf. on Network-Based Information Systems (NBiS)*. IEEE, 2011, pp. 229–235.

[10] M. T. Younis, S. Yang, and 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.

[11] 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*, vol. 61, no. 6, pp. 810–837, 2001.

[12] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000)*. IEEE, 2000, pp. 185–199.

[13] J. Kołodziej and F. Xhafa, "Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1035–1046, 2011.

[14] E. Pacini, C. Mateos, and C. G. Garino, "Distributed job scheduling based on swarm intelligence: A survey," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 252–269, 2014.

[15] S. Selvi and D. Manimegalai, "Multiobjective variable neighborhood search algorithm for scheduling independent jobs on computational grid," *Egyptian Informatics Journal*, vol. 16, no. 2, pp. 199–212, 2015.

[16] S. Selvi and 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)*, vol. 40, no. 3, 2015.

[17] G. Ritchie and J. Levine, "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments," 2004.

[18] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[19] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids," in *Proceedings 8th IEEE Int. Conf. on Advanced Computing and Communications (ADCOM 2000)*, 2000, pp. 45–52.

[20] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[21] F. Xhafa, J. Carretero, L. Barolli, and A. Durresi, "Requirements for an event-based simulation package for grid systems," *Journal of Interconnection Networks*, vol. 8, no. 02, pp. 163–178, 2007.

[22] H. Liu, A. Abraham, and A. E. Hassanien, "Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1336–1343, 2010.

[23] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[24] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi, "Variable neighborhood search: basics and variants," *EURO Journal on Computational Optimization*, vol. 5, no. 3, pp. 423–454, 2017.

[25] E. Alba and G. Luque, "A new local search algorithm for the dna fragment assembly problem," *Evolutionary Computation in Combinatorial Optimization*, pp. 1–12, 2007.

[26] A. B. Ali, G. Luque, E. Alba, and K. E. Melkemi, "An improved problem aware local search algorithm for the dna fragment assembly problem," *Soft Computing*, vol. 21, no. 7, pp. 1709–1720, 2017.

[27] S. Nesmachnow, E. Alba, and H. Cancela, "Scheduling in heterogeneous computing and grid environments using a parallel chc evolutionary algorithm," *Computational Intelligence*, vol. 28, no. 2, pp. 131–155, 2012.

[28] G. F. Minetti, G. Luque, and E. Alba, "The problem aware local search algorithm: an efficient technique for permutation-based problems," *Soft Computing*, pp. 1–14, 2017.

[29] M. Dorigo, M. Birattari *et al.*, "Swarm intelligence." *Scholarpedia*, vol. 2, no. 9, p. 1462, 2007.

[30] F. Xhafa, E. Alba, B. Dorronsoro, B. Duran, and 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.

[31] F. Xhafa, "A hybrid evolutionary heuristic for job scheduling on computational grids," in *Hybrid Evolutionary Algorithms*, Springer, 2007, pp. 269–311.