6-2014

Evaluation of Particle Swarm Optimization Applied to Grid Scheduling

Wilson Higashino
Western University, whigashi@uwo.ca

Miriam A M Capretz
Western University, mcapretz@uwo.ca

M. Beatriz F. Toledo
Universidade Estadual de Campinas, beatriz@ic.unicamp.br

Evaluation of Particle Swarm Optimization Applied to Grid Scheduling

Wilson A. Higashino*†, Miriam A. M. Capretz*, Maria Beatriz Felgar de Toledo†
*Dep. of Electrical and Computer Engineering, Western University, London, Canada
{whigashi, mcapretz}@uwo.ca
†Instituto de Computação, Univ. Estadual de Campinas, Campinas, Brazil
{wah, beatriz}@ic.unicamp.br

***Abstract*—The problem of scheduling independent users' jobs to resources in Grid Computing systems is of paramount importance. This problem is known to be NP-hard, and many techniques have been proposed to solve it, such as heuristics, genetic algorithms (GA), and, more recently, particle swarm optimization (PSO). This article aims to use PSO to solve grid scheduling problems, and compare it with other techniques. It is shown that many often-overlooked implementation details can have a huge impact on the performance of the method. In addition, experiments also show that the PSO has a tendency to stagnate around local minima in high-dimensional input problems. Therefore, this work also proposes a novel hybrid PSO-GA method that aims to increase swarm diversity when a stagnation condition is detected. The method is evaluated and compared with other PSO formulations; the results show that the new method can successfully improve the scheduling solution.**

Keywords*—*Particle Swarm Optimization, Grid Scheduling, Grid Computing, Genetic Algorithms.

I. Introduction

Grid computing is a distributed system paradigm in which multiple computer resources are federated to achieve a common goal. Grids are usually highly heterogeneous and loosely coupled environments, with the constituent resources distributed across many different organizations and administrative domains.

Computational grids focus on sharing processing capabilities to accomplish highly demanding computational tasks. It is in this context that grid scheduling emerges as a problem. Simply stated, grid scheduling refers to the problem of allocating jobs to grid resources. There is a whole family of grid scheduling problems that depend on varying grid and jobs characteristics [1]. This article focuses on the problem of centralized scheduling of independent jobs in a static environment.

The problem of scheduling jobs over heterogeneous resources is known to be NP-hard [2]. Therefore, many heuristic and meta-heuristic techniques have been used to solve it, such as genetic algorithms (GA) [3] [4] and particle swarm optimization (PSO) [5] [6].

This research was intended to carry out a deep study of PSO as applied to the grid scheduling problem and to propose a hybrid PSO-GA algorithm that significantly improves the results of basic PSO. First, an extensive set of experiments was executed, in which often-overlooked implementation details of PSO were analysed. It has been shown that these details can have a huge impact on PSO performance. Next, the rationale for creating the hybrid method is presented, along with experimental results that confirm its ability to find good solutions. At each step, the PSO implementations are compared with genetic algorithms and other well-known heuristics.

This article is organized as follows: Section II reviews related research. In Section III, the grid scheduling problem is formalized, and the basic formulations of the GA and PSO methods are presented. In Section IV, the hybrid PSO-GA algorithm is described, and experimental results are shown in Section V. Finally, Section VI draws conclusions and presents directions for future research.

II. Related Works

Xhafa and Abraham [1] have presented an overview of the grid scheduling problem and the many heuristics that have been used to solve it. Genetic algorithms (GA) [7] and particle swarm optimization (PSO) [8] are both population-based methods that have been applied to this problem.

Genetic algorithms (GAs) are meta-heuristics that use analogies with genetic biological processes to solve optimization problems. Xhafa *et al.* [4] performed a comprehensive study of many different GA formulations and their performance when applied to grid scheduling. Abraham *et al.* [3] presented a grid scheduling method which hybridizes GA with simulated annealing and tabu search. Xhafa *et al.* [9] focused on the struggle GA, which uses a different individual replacement strategy than the traditional GA.

Particle swarm optimization (PSO) is an evolutionary computing technique introduced by Kennedy and Eberhart [8]. Laskari *et al.* [10] have shown that an integer version of PSO can achieve better results than the branch and bound technique when applied to certain optimization problems. Bu *et al.* [5] used this integer PSO to solve the grid scheduling problem, and Zhang *et al.* [11] used a similar approach, but transforming the real solution to an integer using the SPV (smallest position value) [12] method. Finally, Liu *et al.* [6] modelled the scheduling solution as a fuzzy matrix representing the position and velocity of the particles in the PSO algorithm.

Finally, various authors have also explored hybrid meta-heuristics in the grid scheduling context. For example, Chen [13] hybridized PSO with the simulated annealing technique, and Sarathchandar *et al.* [14] used the concept of digital pheromones to improve PSO results.

http://dx.doi.org/10.1109/WETICE.2014.26

III. Background

This section presents the classical formulation of the grid scheduling problem and the main methods used to solve it.

A. Grid Scheduling

The classical formulation of the grid scheduling problem considers J_i jobs, $i \in \{1, 2, \ldots, m\}$, and G_j grid nodes (resources), $j \in \{1, 2, \ldots, n\}$. The length of each job, measured in cycles, and the speed of each resource, measured in cycles/second, are known. The following notation is also defined:

- C_{ij}, the time when resource j finishes processing job i; $C_{ij} = 0$ if the job i does not execute on j;
- ΣC_j, the time when resource j finishes processing all jobs allocated to it.

Using this notation, grid scheduling can be defined as the problem of allocating jobs to resources to minimize an optimization criterion. The $makespan$, defined as $makespan = max(\Sigma C_j)$, $1 < j < m$, represents the time when the last job is finished and is one of the most often used criteria.

Ali *et al.* [15] defined the same problem in terms of an ETC (estimated time to complete) matrix. In this formulation, each position $ETC(i, j)$ of the matrix represents the time that resource j needs to process job i. They expressed the completion time of a resource j as:

$$completion(j) = ready(j) + \sum_{\{i \in Jobs(j)\}} ETC(i, j) \quad (1)$$

where $ready(j)$ is the time at which resource j is available and $Jobs(j)$ is the set of jobs allocated to the resource. Given this equation, $makespan$ can be defined as:

$$makespan = max\{completion(j) | j \in \{1, 2, \ldots, n\}\} \quad (2)$$

The ETC matrix can represent more general problems in which the fact that a resource m_1 runs a job i faster than j does not imply the same for another resource m_2. Ali *et al.* [15] presented a method to generate ETC matrices that can vary according to three criteria: job heterogeneity, machine heterogeneity, and the matrix consistency. In a consistent matrix, if a resource m_i processes a job faster than another resource m_j, the same will hold for all other jobs. The ETC formulation is used in this article because of its broad applicability.

B. Genetic Algorithms

Genetic algorithms (GAs) are meta-heuristics that use analogies with genetic biological processes to solve optimization problems. GAs are implemented as iterative algorithms that use a population of individuals to represent possible solutions to the problem at hand. In each iteration, the best individuals of the population originate an offspring through crossover and mutation operators. These new individuals, along with some of their parents, form a new population that is used in the next iteration. This process is repeated until a good solution is found or a specified number of iterations have been executed. Figure 1 shows the generic framework of a GA.

```
1:  Initialization: Create initial population
2:  while (not stop condition) do
3:      Fitness: Compute fitness values for the population
4:      Scale: Scale the computed values
5:      Produces the next population by the following steps
6:      Selection: Select parents individuals
7:      Elite: The elite parents survive to the next generation
8:      Crossover: Produce $P_c$% of the children by crossover
9:      Mutation: Produce $(1 - P_c)$% of the children by mutation
10: end while
```

Fig. 1. A generic Genetic Algorithm

One of the most important aspects of a GA is how an individual of the population encodes a possible solution to the problem. GA implementations may also vary according to how the population is initialized and how the selection, crossover, and mutation steps are executed. Therefore, the generic GA framework can originate many different specific algorithms by using different individual representations and different selection, crossover, and mutation operators.

C. Particle Swarm Optimization

Particle swarm optimization (PSO), introduced by Kennedy and Eberhart [8], is an algorithm that tries to simulate the behaviour of swarms present in nature, such as flocks of birds and schools of fish, to solve optimization problems. In this algorithm, information is shared among individuals called particles. Each particle represents a possible solution and moves around in the D-dimensional search space with a velocity determined by a combination of its knowledge of the best solution with the knowledge of other members of the swarm. Formally, each particle i of the swarm has three D-dimensional vectors associated with it:

- $\vec{x}_i(t)$ - Position of particle i at time t;
- $\vec{v}_i(t)$ - Velocity of particle i at time t;
- $\vec{p}_i$ - Previous best position of particle i.

The particle also tracks the value found in its best position using the variable $pbest_i$. In each iteration, the particle updates its position using the following formula:

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \quad (3)$$

while the velocity is updated using:

$$\begin{aligned}\vec{v}_i(t) = \omega \cdot \vec{v}_i(t-1) &+ \varphi_1 \cdot rand_1 \cdot (\vec{p}_i - \vec{x}_i(t-1)) \\ &+ \varphi_2 \cdot rand_2 \cdot (\vec{p}_g - \vec{x}_i(t-1)) \quad (4)\end{aligned}$$

In this equation, ω, φ_1, and φ_2 are algorithm parameters, $rand_1$ and $rand_2$ are random numbers taken from a uniform distribution in the range $[0, 1]$, and $\vec{p}_g$ is the best position found so far by the whole swarm. The ω parameter is called the inertia weight, and can be interpreted as the fluidity of the medium in which the particles move. A high ω value represents a "low-friction" environment suitable for space exploration, while a low value represents a "high-friction" environment better for space exploitation (local-oriented search). The parameters φ_1 and φ_2, on the other hand, represents the attraction towards the particle's own best position and the best position of the group, respectively. Another important observation is that an unbounded velocity can lead to a swarm "explosion". Therefore, many authors bound the velocity within $[-V_{max}, +V_{max}]$, where V_{max} is usually set to a fraction of the maximum $\vec{x}$.

D. Heuristics

Many heuristics have been used for grid scheduling, and their relative performance has been compared in different studies [16] [17]. This paper uses the *LJFR-SJFR* and *min-min* heuristics because of their simplicity and the good results achieved in these comparisons.

The *LJFR-SJFR* (largest job on fastest resource-shortest job on fastest resource) heuristic works as follows:

1) Jobs are sorted in descending order according to size. Resources are sorted in descending order according to speed;
2) The first n jobs are allocated to the corresponding first n resources (LJFR heuristic);
3) For the remaining jobs, the following is repeated:
    a) The shortest remaining job is allocated to the first available resource (SJFR);
    b) The largest remaining job is allocated to the next available resource (LJFR).

The *min-min* heuristic, on the other hand, is based on the idea of scheduling first the task that finishes the earliest. The following steps make up the algorithm:

1) Given the set U of unscheduled jobs, the minimum completion time $M(i)$ is calculated for each job $i \in U$;
2) The job with the smallest $M(i)$ is allocated to the corresponding resource and is removed from U;
3) These two steps are repeated until no more jobs exist.

IV. Hybrid PSO-GA

The hybrid PSO-GA (**H_PSO**) algorithm presented in this paper is based on two main modifications to the original PSO method. First, **H_PSO** creates elite particles in the initial swarm. Second, **H_PSO** detects situations in which the swarm gets stuck at a local minimum and tries to add diversity by executing a complete iteration of a genetic algorithm. Figure 2 shows a complete description of this algorithm.

The first three lines constitute the initialization phase, in which two particles are generated using the *min-min* and *LJFR-SJFR* heuristics and the others are randomly initialized. The body of the algorithm remains essentially the same as the basic PSO, except for the fact that, if the best global value ($gbest$) is not updated for T_r iterations, then an iteration of the GA is executed. A description of the $GA\_Iteration$ function can be seen in Figure 3. In this listing, N is the number of particles being used. The function introduces three new parameters to **H_PSO**: E_c, which represents the number of elite particles that will be carried over to the next iteration; C_f, which represents the percentage of the swarm that will be generated using the crossover operator; and T_s, which represents the size of the tournament used by the selection operator. Details of the GA operators used can be found in Xhafa *et al.* [4].

```
1: $i_1 \leftarrow$ Min-Min heuristic
2: $i_2 \leftarrow$ LJFR-SJFR heuristic
3: $i_3, \cdots, i_N \leftarrow$ init particles with random $\vec{x}_i$ and $\vec{v}_i$
4: while not stop condition do
5:   for all particles $i$ do
6:     $current_i \leftarrow f(\vec{x}_i)$
7:     if $current_i < pbest_i$ then
8:       $\vec{p}_i \leftarrow \vec{x}_i$
9:       $pbest_i \leftarrow current_i$
10:    end if
11:  end for
12:  $\vec{p}_g \leftarrow$ best position found so far
13:  $gbest \leftarrow$ best fitness value
14:  if $gbest$ is unchanged for $T_r$ iterations then
15:    $GA\_Iteration()$
16:  else
17:    for all particles $i$ do
18:      Update $\vec{x}_i$ and $\vec{v}_i$ according to Eqs. 3 and 4
19:    end for
20:  end if
21: end while
```

Fig. 2. Hybrid PSO-GA algorithm

```
1: sort particles $i$ according to $pbest_i$
2: Elite: select the best $E_c$ particles to the next swarm
3: $c_{count} \leftarrow C_f * N$
4: $m_{count} \leftarrow N - c_{count} - E_c$
5: Select: select $c_{count} + m_{count}$ particles using tournament
   of size $T_s$
6: Crossover: generate $c_{count}$ particles using cycle crossover
7: Mutation: generate $m_{count}$ particles using rebalancing
```

Fig. 3. $GA\_Iteration()$ function

V. Experiments

This section describes the experiments carried out to evaluate the PSO algorithm as applied to the grid scheduling problem and how the hybrid PSO method presented in Section IV improves the results of the base PSO.

The experiments are divided into three sets. The first set of experiments aims to validate the GA and PSO implementations and to compare them with well-known heuristics. In the second set of experiments, the base PSO is incrementally modified towards the hybrid PSO, and the final results of the method are presented. Finally, in the last set of experiments, the hybrid PSO is compared with the Liu *et al.* [6] PSO formulation.

A. Validation

In the first set of experiments, six different input instances of the grid scheduling problem were used. The first two instances were defined using the model that specifies the length of the jobs in J and the speed of the resources in G and can be seen in Table I. The other four instances are ETC matrices generated using the method by Ali *et al.* [15], and are described in Table II. Column m represents the number of jobs and column n the number of resources in the input instance. All algorithms were evaluated according to the $makespan$ criterion, as defined in Equation 2.

TABLE I. SIMPLE INPUT INSTANCES.

Instance 1	$J = \{19, 23, 24, 20, 20, 25, 16, 21, 24, 15\}$ $G = \{4, 3, 2\}$
Instance 2	$J = \{6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, 60\}$ $G = \{4, 3, 2\}$

TABLE II. ETC MATRIX INPUT INSTANCES.

Instance	**Consistent**	**Job het.**	**Machine heterog.**	**m**	**n**
3	Yes	Low	Low	64	8
4	Yes	Low	High	64	8
5	Yes	High	High	64	8
6	Yes	High	Low	64	8

Table III shows the results of these experiments. The second and third columns are the results of two heuristics used as a basis for comparison: *LJFR-SJFR* and *min-min*. The fourth column presents the average of ten runs of the GA for each input instance. The GA implementation used the parameters and operators with the best results in Xhafa *et al.* [4], as shown in Table IV.

The PSO results are shown in the fifth column (**B_PSO**) of Table III. The basic PSO implementation used in these experiments represents particle velocity as a vector of floating-point numbers within $[-V_{max}, +V_{max}]$. The functions $rand_1$ and $rand_2$ generate a vector of random numbers within $[0, 1]$ of size m, and each dimension of this vector multiplies its corresponding dimension of the position differences in Equation 4. If after velocity calculations, any velocity component is larger (smaller) than $+V_{max}$ ($-V_{max}$), then the component is truncated to $+V_{max}$ ($-V_{max}$). Similarly, particle positions are stored as a vector of floating-point numbers within $[1, (X_{max} + 1)[$. The algorithm uses a rounded-down version of particle position to calculate the $makespan$ value.

The values shown in Table III are the average $makespan$ obtained by ten runs of all top ten PSO parameter combinations. In order to select the best parameter combinations for the algorithm, a separate experiment was executed in which all combinations of the parameters described in Table V were evaluated. The ten combinations with best overall results are shown in Table VI. For all combinations, the value of ω_{max} was set to 0.9, and the number of iterations was 400.

From the results shown in Tables VI and III, it can be concluded that:

- The *min-min* heuristic has surprisingly good results for the larger and more complex input instances, with the best performance on three of the four ETC matrices. For the simpler instances, both GA and PSO achieved results very close to the optimal solution (23.25 and 46 respectively). However, their performance on the ETC matrices might not justify their use because they have a higher computational cost than the heuristics.
- In PSO, the use of a decreasing ω is very important. Indeed, all the top parameter combinations have $\omega_{min} = 0.1$ or $\omega_{min} = 0.4$. This variation has the effect of transitioning the PSO population from an exploration phase, when ω is high, to an exploitation phase, when ω is low and the particles fine-tune solutions found in the first phase;
- Almost all PSO parameter combinations have $\varphi_1 = 2$ and $\varphi_2 = 1.49$, which means that when particles consider their own "self-knowledge" more important than "social-knowledge", the results tend to be better.

B. PSO improvements

The experiments of Section V-A showed that the **B_PSO** results were worse than the GA and the heuristics for most input instances. In order to understand and improve the algorithm performance, modifications to the base PSO were proposed and tested. In this section, these modifications and their experimental results are shown.

1) Limiting behaviour: an important implementation detail of the PSO algorithm is the behaviour of particles when they reach the boundaries of the region that encompasses the solution space. This discussion is often omitted from PSO algorithm descriptions, but the following experiment shows that it has important effects on the final results.

In the modification proposed, position dimensions that exceed X_{max} or X_{min} are truncated to X_{max} or X_{min}, and the corresponding velocity dimensions are zeroed. The idea is to simulate a boundary that absorbs particle movement. The results of this version are also shown in Table III in the column **A_PSO** (Absorption PSO). Once again, for each input instance the algorithm was executed ten times for all parameter combinations shown in Table VI. Each cell shows the average $makespan$ of all these executions and the improvement over the **B_PSO**. Note that the **A_PSO** version has better performance than **B_PSO** for all input instances. Because of these results, the absorption behaviour was incorporated into the final PSO algorithm presented in this article.

2) Initial Particle: based on the good results of the *min-min* heuristic in the solution of Instances 3 to 6, the **A_PSO** implementation was modified to use the result of the *min-min* heuristic as one of its initial particle. The results of this experiment can be seen in the column **M_PSO** of Table III. Note that **M_PSO** has significantly better results than the others for Instances 3 to 6 and only slightly worse results than GA in the first two instances.

To assess **M_PSO** performance further, four more input instances were generated according to the ETC matrix model. These instances are described in Table VII. The same combinations of job and machine heterogeneity were used, but with larger values of m and n representing higher dimensional (and harder) problems. The results of the execution of the **M_PSO**, GA, *min-min*, and **A_PSO** algorithms for these new instances can be seen in Table VIII. Note that even though the **M_PSO** algorithm has much better results than the GA and **A_PSO**, the improvement over the *min-min* heuristic is small.

The **M_PSO** algorithm was further analysed to understand its poor performance for these inputs. The implementation was instrumented to calculate two metrics: 1) population $diversity$ for each iteration, as proposed by Riget and Vesterstrom [18]; 2) The swarm $centerchange$, calculated as the Euclidean distance between the center of mass of two consecutive swarms. Figure 4 shows the evolution of the $diversity$ and $centerchange$ metrics obtained from a typical **M_PSO** run. Note how both metrics quickly drop at the beginning of the run, especially the $centerchange$ values.

TABLE III. PSO VALIDATION EXPERIMENTS.

Instance	LJFR-SJFR	Min-Min	GA	B_PSO	A_PSO		M_PSO	
1	24.25	25.75	23.26	23.32	23.31	-0.04%	23.31	-0.04%
2	48.00	56.00	46.22	46.34	46.25	-0.19%	46.24	-0.22%
3	1955.86	1652.79	1751.18	1831.33	1791.32	-2.18%	1538.05	-16.01%
4	114263.94	94816.27	101348.19	108373.81	103943.33	-4.09%	90433.92	-16.55%
5	3516259.74	3270218.66	3154809.88	3283008.74	3224763.15	-1.77%	2732893.89	-16.76%
6	51502.87	46706.86	47575.73	50695.48	50410.18	-0.56%	45403.94	-10.44%

TABLE IV. GENETIC ALGORITHM PARAMETERS.

Population size	80
Generations	2500
Initial population	Random generated, one particle from LJFR-SJFR
Selection operator	Tournament
Tournament Size	$T_S = 4$
Crossover operator	Cycle Crossover
Mutation probability	$P_c = 80\%$
Mutation operator	Rebalancing
Stop condition	Number of generations OR 50 generations with less than $1e-6$ change in the average fitness value

TABLE V. PSO PARAMETERS.

Population Size	60, 30
φ_1	2.0, 1.49
φ_2	2.0, 1.49
ω_{max}	0.9
ω_{min}	0.9, 0.4, 0.1
$\mathbf{X_{max}}$	n (number of resources)
$\mathbf{V_{max}}$	$X_{max}, 0.6 * X_{max}, 0.2 * X_{max}$
Iterations	400
Stopping Condition	Number of iterations

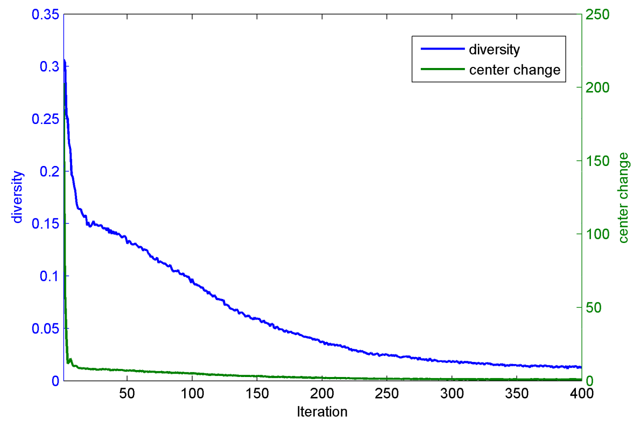

Fig. 4. *diversity* and *centerchange* evolution.

Analysis of the graphic suggests that the swarm quickly converges to a small region of the solution space and that, as the value of ω decreases , this grouping becomes even tighter. In other words, the particles quickly confine themselves to a small region of the search space, probably around the local minima found by the *min-min* heuristic, which was used as one of the initial particles of the algorithm.

3) GA and PSO hybridization: as a result of the observations made for the last experiment, the **H_PSO** algorithm has been proposed as described in Section IV. Table IX shows the results of running the **H_PSO** algorithm using the same input test instances as earlier. The parameters used were $T_r = 50$, $E_c = 5$, $C_f = 0.8$, and $T_S = 4$. The **H_PSO** algorithm exhibited better performance than the others for all the inputs without adding significant burden to the base PSO implementation.

TABLE VI. BEST PSO PARAMETER COMBINATIONS.

#	Pop. Size	ω_{min}	φ_1	φ_2	V_{max}
1	60	0.1	2	1.49	0.6
2	60	0.4	2	1.49	0.6
3	60	0.1	2	1.49	1
4	60	0.1	2	2	0.6
5	30	0.1	2	1.49	0.6
6	60	0.4	2	1.49	1
7	60	0.1	2	1.49	0.2
8	30	0.4	2	1.49	1
9	60	0.4	1.49	1.49	1
10	60	0.4	2	1.49	0.2

TABLE VII. NEW *ETC* MATRIX INPUT INSTANCES.

Instance	Consistent	Job het.	Machine heterog.	m	n
7	Yes	Low	Low	512	16
8	Yes	Low	High	512	16
9	Yes	High	High	512	16
10	Yes	High	Low	512	16

C. Alternative PSO implementation

The experiments described in this section were intended to compare the **H_PSO** implementation with the fuzzy PSO (**F_PSO**) presented in Liu *et al*. [6]. According to their results, the best parameters for **F_PSO** are those presented in Table X. Their experiments were run using $50 * m * n$ iterations, where m is the number of jobs and n is the number of grid resources available to solve the input problem. For the sake of this research, the number of iterations was set to 400, because otherwise the algorithm would spend too much time running and could not be used in a practical scenario. Similarly to the other experiments, the number of particles was set to 60. Note also that Liu *et al*. made no mention of the value used for V_{max}. Therefore, the algorithm was tested with $V_{max} = X_{max}, 0.6{*}X_{max}$, and $0.2{*}X_{max}$. To perform a more equitable comparison, the **F_PSO** algorithm was also modified to generate an initial particle using the *min-min* heuristic.

The results of the algorithm can be seen in the **F_PSO** column of Table IX. Each cell shows the average of running the algorithm 10 times for all three different values of V_{max}. The results for **H_PSO** are very similar to the **F_PSO** results. Nevertheless, the **H_PSO** implementation particle representation uses much less memory than **F_PSO**, and its execution time tends to be shorter because it uses a less complex set of operations. Some preliminary results show that **H_PSO** is typically 40% faster than the **F_PSO** formulation.

VI. CONCLUSION

This article analysed the PSO algorithm as applied to the grid scheduling problem. For purposes of comparison, a genetic algorithm and two heuristics (*LJFR-SJFR* and *min-min*) were also investigated. The results showed that the basic PSO can be successfully used in small problem instances and can provide good results compared to other methods. It was

TABLE VIII. EXPERIMENTS WITH LARGER INPUT INSTANCES.

Instance	M_PSO	GA		Min-Min		A_PSO	
7	5980.7377	7274.28	21.63%	6003.01	0.37%	9973.086	66.75%
8	316694.2394	432036.83	36.42%	319733.75	0.96%	865089.9009	173.16%
9	7811849.476	10908183.59	39.64%	7921706.99	1.41%	23677640.77	203.10%
10	164443.6232	199369.12	21.24%	165088.09	0.39%	266718.9151	62.19%

TABLE IX. HYBRID PSO-GA EXPERIMENTS.

Instance	H_PSO	GA		Min-Min		M_PSO		F_PSO	
1	23.26	23.26	0.00%	25.75	10.70%	23.31	0.19%	23.61	1.49%
2	46.17	46.22	0.10%	56.00	21.28%	46.24	0.15%	46.76	1.27%
3	1521.69	1751.18	15.08%	1652.79	8.62%	1538.05	1.07%	1569.38	3.13%
4	87750.84	101348.19	15.50%	94816.27	8.05%	90433.92	3.06%	86257.74	-1.70%
5	2729859.92	3154809.88	15.57%	3270218.66	19.79%	2732893.89	0.11%	2800914.65	2.60%
6	43797.76	47575.73	8.63%	46706.86	6.64%	45403.94	3.67%	43783.93	-0.03%
7	5868.24	7274.28	23.96%	6003.01	2.30%	5980.7377	1.92%	5876.54	0.14%
8	306009.87	432036.83	41.18%	319733.75	4.48%	316694.2394	3.49%	305011.48	-0.33%
9	7551325.00	10908183.59	44.45%	7921706.99	4.90%	7811849.476	3.45%	7502353.66	-0.65%
10	160234.17	199369.12	24.42%	165088.09	3.03%	164443.6232	2.63%	159958.83	-0.17%

TABLE X. FUZZY PSO PARAMETERS.

Population Size	20
φ_1	1.49
φ_2	1.49
ω_{max}	0.9
ω_{min}	0.1
Iterations	$50 * m * n$

also shown that some often-overlooked implementation details, such as the behaviour of particles at the boundaries of the search space, have significant impact on the performance of this method.

In addition, this research also studied the PSO method on larger problems composed of a larger number of jobs and resources. For this set of inputs, the basic PSO showed some limitations and a high probability of early convergence to local minima. To tackle these problems, a new hybrid PSO-GA algorithm was proposed based on the hybridization of PSO with GA. **H_PSO** was evaluated and provided significant improvement over the basic version of PSO.

As for future work, the authors plan to study alternative representations for swarm particles and to compare the proposed method with other PSO variations for discrete and optimization problems. Additionally, a more comprehensive speed and memory consumption comparison will be carried out to confirm the initial findings presented in this article.

REFERENCES

[1] 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, Apr. 2010.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[3] A. Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids," in *IEEE International Conference on Advanced Computing and Communications*, 2000, pp. 45–52.

[4] F. Xhafa, J. Carretero, and A. Abraham, "Genetic Algorithm Based Schedulers for Grid Computing Systems," *International Journal of Innovative Computing, Information and Control*, vol. 3, no. 5, pp. 1–19, 2007.

[5] Y.-p. Bu, W. Zhou, and J.-s. Yu, "An Improved PSO Algorithm and its Application to Grid Scheduling Problem," *2008 International Symposium on Computer Science and Computational Technology*, pp. 352–355, 2008.

[6] 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, Oct. 2010.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[9] F. Xhafa, B. Duran, A. Abraham, and K. P. Dahal, "Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids," in *7th Computer Information Systems and Industrial Management Applications*. IEEE, Jun. 2008, pp. 275–280.

[10] E. Laskari, K. Parsopoulos, and M. Vrahatis, "Particle swarm optimization for integer programming," *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 2, pp. 1582–1587, 2002.

[11] L. Zhang, Y. Chen, R. Sun, and B. Yang, "A Task Scheduling Algorithm Based on PSO for Grid Computing," *International Journal of Computational Intelligence Research*, vol. 4, no. 1, 2008.

[12] M. F. Tasgetiren, Y.-c. Liang, M. Sevkli, and G. Gencyilmaz, "Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem," *International Journal of Production Research*, vol. 44, no. 22, pp. 4737–4754, Nov. 2006.

[13] R.-m. Chen, "Application of Discrete Particle Swarm Optimization for Grid Task Scheduling Problem," in *Advances in Grid Computing*, Z. Constantinescu, Ed. InTech, 2011.

[14] A. P. Sarathchandar, V. Priyesh, and D. D. H. Miriam, "Grid Scheduling using Improved Particle Swarm Optimization with Digital Pheromones," *International Journal of Scientific & Engineering Research*, vol. 3, no. 6, pp. 1–6, 2012.

[15] S. Ali, H. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*. IEEE Comput. Soc, 2000, pp. 185–199.

[16] A. K. Bardsiri and S. M. Hashemi, "A Comparative Study on Seven Static Mapping Heuristics for Grid Scheduling Problem," *International Journal of Software Engineering and Its Applications*, vol. 6, no. 4, pp. 247–256, 2012.

[17] 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, and R. F. Freund, "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, Jun. 2001.

[18] J. Riget and J. S. Vesterstrom, "A diversity-guided particle swarm optimizer-the ARPSO," Aarhus Universitet, Aarhus, Tech. Rep., 2002.