2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Hybrid Discrete Particle Swarm Optimisation Method for Grid Computation Scheduling

Stephen Bennett. Su Nguyen, and Mengjie Zhang
Evolutionary Computation Research Group
Victoria University of Wellington, PO Box 600, Wellington, New Zealand
bennetstep@myvuw.ac.nz, {su.nguyen,mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Allocating jobs to heterogeneous machines in grid systems is an important task in computational grid to effectively utilise computational resources. Particle swarm optimisation (PSO) has been recently applied to grid computation scheduling (GCS) problems and shown very promising results as compared to other meta-heuristics in the literature. However, PSO with the traditional position updating mechanism still has problem coping with the discrete nature of GCS. This paper proposed a new updating mechanism for discrete PSO that directly utilise discrete solutions from personal and global best particles. A new local search heuristic has also been proposed to refine solutions found by PSO. The results show that the hybrid PSO is more effective than other existing PSO methods in the literature when tested on two benchmark datasets. The hybrid method is also very efficient, which makes it suitable to deal with large-scale problem instances.**

***Index Terms*—particle swarm optimisation, grid computing, scheduling, local search**

I. Introduction

A computational grid (CG) is a distributed computing system that includes a large and heterogeneous computing resources [1], [2], [3], [4]. CG is considered an effective approach to dealing with large-scale distributed real-world applications [5]. Scheduling in CG environments (i.e. allocation of jobs to grid computing resources) is important and is a hard computational task even when there is no dependencies among jobs [2] because of different real-world requirements such as heterogeneity of jobs, multiple conflicting objectives, and uncertainty.

This paper focuses on grid computation scheduling (GCS) with independent jobs. Given a number of jobs (or tasks) n and a number of machines m, the objective is to find the optimal solution to allocate jobs to machines in order to minimise the makespan $C_{max} = \max\{C_i\}$, where C_i is the completion time of machine $i = 1 \dots m$. It is noted that a job can only be processed on one machine and a machine can only process one job at a time. There are no interdependence between jobs and preemption is not allowed. Different from traditional parallel machine scheduling problems, GCS includes a set of heterogeneous machines and a specific job j will have different processing times p_{ji} at different machines j. This problem has been shown to be NP-Hard [6], [7] and often involves a large number of jobs and machines (resources); therefore, finding optimal solutions with exact methods are impractical. For this reason, heuristics and meta-heuristics have become a practical approach to GCS to find satisfactory solutions for real size problems.

Braun et al. [8] compared eleven heuristics for GCS with independent jobs. In their paper, a large dataset was also developed based on expected time to compute (ETC) matrix to evaluate the performance of proposed heuristics. From their experiments, genetic algorithm (GA) provided the best performance in most cases and the min-min heuristic (which step-by-step assigned job to machine that results in the earliest completion time) is a good approach to quickly generating a solution with a reasonably short makespan. Page and Naughton [9] proposed another GA method for GCS that uses a list scheduling heuristic to create a good randomised initial population. Specialised mutation operators were also proposed to improve the performance GA. Ritchie and Levine [10] developed a hybrid ant colony optimisation (ACO) for grid scheduling. Tabu search [11] was used within the proposed method to refine solutions obtained by ACO. The hybrid ACO is shown to be better than GA methods and other heuristics. A cellular memetic algorithm (cMA) was developed by Xhafa et al. [12] to minimise both makespan and flowtime simultaneously. Different local heuristics were also examined in this paper. The results show that cMA is an effective and efficient approach to GCS. Xhafa et al. [2] developed a new tabu search method for GCS. Different specialised diversification strategies have also been considered in the proposed tabu search method to enhance its effectiveness. The proposed method is significantly faster than other heuristics in the literature such as tabu search [11], hybrid ACO [10], and cMA [12] while providing very good solutions. Other heuristics have also been proposed for GCS such as simulated annealing [8], [13], struggle genetic algorithm [14], sufferage [15]. hybrid GA [16].

Particle Swarm Optimisation (PSO) [17], [18] is a swarm intelligence technique based on simulated social behaviours. In PSO, a swarm or population of particles (candidate solutions) moves in the solution search space. Each solution is usually encoded as a vector of real numbers that is treated as the position of a particle in the swarm. Each particle/solution is assigned a fitness value based on its performance (quality of the solution). Then particles update their positions by being accelerated towards referenced particles such as global best, and local best (based on certain topologies). The key idea is that this movement will help guide the swarm towards the

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

global optimal solutions in the search space.

PSO has been applied to solve different hard optimisation problem and shown very promising results [19], [20], [21], [22], [23], [24], [25]. Recently, some PSO methods [1], [26] have been proposed to deal with GCS problems with independent jobs. Liu et al. [1] proposed a continuous PSO (CPSO) method for GCS. In this method, both a particle's position and velocity are represented as real numbered matrices ($n \times m$). In order to construct a schedule the position matrix must first be converted into a schedule by assigning each job to the machine that has the highest (normalised) value in the corresponding column of that job in the position matrix. The position matrix in this method is considered as a fuzzy matrix in which an element represents the degree of membership of the corresponding job and machine. The experiments showed that CPSO is better than SA and GA. Izakian et al. [26] proposed a discrete PSO approach (DPSO) for grid scheduling. In their method, particles are represented with a real number velocity matrix of jobs by machines. The value at a given job/machine position in the matrix specifies how much that job is associated with that machine. The position of a particle is an array of integers, where each position in the list is a job and the integer at that position is the machine the job is assigned to. In the particle updating phase a particle's velocity is changed to be more like the global best particle's velocity and its own personal best velocity based on the social-recognition and self-recognition components, respectively. Once the velocity is updated, the particle's position is set so that each job is assigned to the machine that has the highest value in that job's column of the velocity matrix. Their experiments showed that DPSO is better than other heuristics such as min-min, GA, and hybrid ACO [10].

Although the results showed that PSO is very competitive as compared to other meta-heuristics in the literature [8], [10], [12], there are still some limitations with the existing PSO methods for GCS. The first is that it is computationally expensive to repeatedly convert between continuous and discrete representations [26] and using a matrix makes the cost of updating a particle $O(n \times m)$. This is particularly a problem for the CPSO as it has to perform a number of matrix additions, subtractions and multiplications and then afterwards convert the result into a discrete schedule for fitness evaluation. The second limitation is that a continuous representation is not an ideal approach to a discrete optimisation problem such as GCS. This is true for both CPSO and the real numbered velocity matrix of DPSO. For example, a particle can be significantly altered without influencing significantly the schedule its representing, which makes it easier for a particle to become trapped at a local optimum. Small changes, even a lot of small changes on different dimensions that add up to quite a large change, may have no impact on a particle's fitness. This limits the exploration ability of such algorithms and increases the number of iterations required. Additionally, modelling the solution as a job/machine matrix significantly increases the dimensionality as well as the complexity of the space the algorithms. The third limitation is that the existing PSO methods do not incorporate any knowledge of the problem domain during the search. As a result, PSO can spend a lot of time searching for good solutions, especially for large-scale problem instances usually encountered in GCS.

A. Goals

In order to overcome these limitations, a new discrete PSO method is proposed in this study. In this proposed method, a new position updating mechanism and a new local search heuristic are developed in order to improve both exploration and exploitation abilities while reducing the computational costs of of PSO. Three research objectives of this paper are:

1) Develop a new position updating mechanism for the discrete representation in PSO.
2) Develop a new efficient local search heuristic to refine solutions obtained by PSO.
3) Compare the proposed PSO against other existing PSO methods and analyse the behaviours of these methods.

B. Organisation

The rest of this paper is organised as follows. The next section proposes the new discrete PSO (NDPSO) method for GCS with independent jobs. In Section III, the parameter settings for the proposed PSO and the datasets used for our experiments are described. The performance of NDPSO is compared to that of the existing PSO methods in Section IV. Then, we analyse the behaviours of NDPSO to understand its effectiveness. Finally, we provide conclusions and discussions for future research in Section V.

II. Proposed method

In this section, we will describe the discrete representation used for GCS. Then we showed a new approach to position updating and the new local search heuristic for refining solution obtained by PSO. Finally, the overall algorithm for NDPSO is presented.

A. Representation

In the proposed NDPSO, a particle's position $\mathbf{x} = (x_1, \ldots, x_n)$ is a one-dimensional array of integers, where each element x_j in the array represents the machine assignment of a specific job j, which is the index of a machine used to process job j. The dimension of the particle's position is the number of job n and each element in the array can be the indices of any machines that are able to process the corresponding job (in this paper, a job can be processed at any machine). An example solution of GCS is shown in Figure 1. In this case, the **first** element x_1 is **2** indicates that job **1** will be processed by machine **2**, as shown in the gantt chart, with the processing time p_{12}. Because there is no job interdependence and makespan is used as the objective, a GCS solution here does not need to consider the sequence of jobs at each machine. Different from the representations used in CPSO [1], the position of particle in NDPSO directly represents all job/machine assignments rather than indirectly reflect the relation between jobs and machines through position

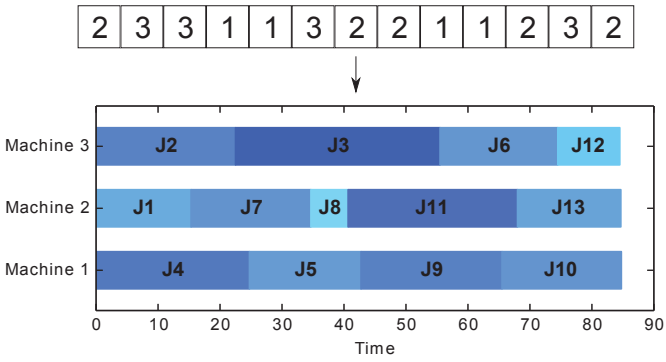

Figure 1: Representation of a GCS solution in NDPSO.

and velocity matrices. This allows GCS solutions represented in a more compact way while still covering all possible assignments. Another advantage of this representation is that a particle's position can always be decoded to a feasible solution. This representation is the same as the one used in DPSO [26].

B. Position updating mechanism

Given that a particle's position is an array of integer numbers, the traditional updating mechanism (through vector addition, subtractions and multiplication) can still be used to update the particle's position. However, this approach is not natural in GCS and does not utilise effectively the information from *personal best* (the best position found by a particle) and *global best* (the best position found by the swarm) [17]. Increasing or decreasing the elements in a particle's positions does not really help particles move towards better positions because there is no correlation between the indices and other characteristics of jobs. The updating mechanism in DPSO can handle the discrete position directly but it is still governed by the traditional updating mechanism, which can be time consuming when dealing with large instances. In this paper, we proposed a new updating mechanism that is more suitable for the discrete representation.

For each job to machine assignment x_j in the particle's position $\mathbf{x}$, a random number r_1 from $[0, 1]$ is generated. If r_1 is greater than or equal to a predefined updating probability u, the corresponding assignment x_j will not be updated. Otherwise, a second random number r_2 from $[0, 1]$ is generated. If r_2 is less than the parameter p the job to machine assignment x_j is replaced with the corresponding assignment of the particle's personal best. If r_2 is above p then x_j is replaced with the corresponding assignment from the global best. This process is demonstrated in Figure 2, where the highlighted machine assignments from the global best, personal best and current position are combined to create a new particle position. Lastly, a third random float r_3 from $[0, 1]$ is generated and if r_3 below the mutation rate m then the job assignment is replaced with a random machine index.

In NDPSO, u and p are two important parameters that determine how global best and personal best particles can be used to update the current position of a particle. When u is small, the particle will be less likely to change its

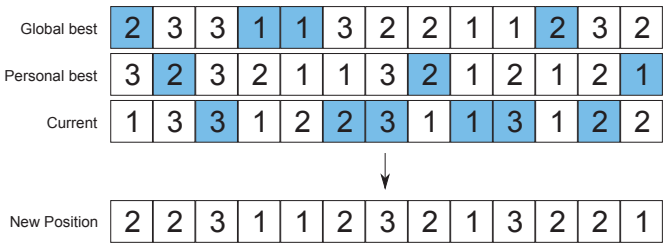

Figure 2: Updating mechanism in NDPSO.

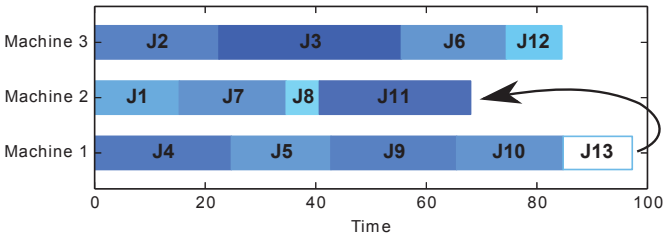

Figure 3: Proposed local search heuristic.

position. When u is higher, the position will be more likely to be updated based on positions of global best and personal best particles. Meanwhile, the parameter p controls which information from global best and personal best particles are preferred to updating the current position of a particle. If $p = 0$, only information from the global best particle is used to update the current position $\mathbf{x}$. A higher p will indicate that it is more likely to use the personal best position to update $\mathbf{x}$. Because the velocities in conventional PSO is not really useful in the discrete representation here, we do not use them in NDPSO to avoid extra computational costs. Although this updating mechanism is significantly different from traditional approach, it is still based on the use of *cognitive* and *social* experience in the swarm; and it is basically a simplified way to simulate swarm behaviour in a discrete search space. A mutation is also applied in this case in order to improve the diversity of the proposed method. In general, the new updating mechanism is faster than the updating mechanisms used in previous work because there is less work involved in updating a particle (i.e. $O(n)$).

C. Local search heuristic

As mentioned, GCS usually involves many different jobs and machines and it is very unlikely that good solutions can be found from random initialisation. Therefore, searching for (near) optimal solutions through PSO will be difficult. For this reason, it would be useful to apply an efficient local heuristic to quickly guide the swarm towards more promising solutions. A new local search heuristic is proposed in this paper to help refine the GCS solutions found by NDPSO without significantly increase the computational times. The pseudo code of the local search heuristic is presented in Algorithm 1.

In this algorithm, $f(\mathbf{x})$ is the makespan obtained from solution $\mathbf{x}$. The $max\_machine(\mathbf{x})$ (or $min\_machine(\mathbf{x})$) function gives the index of the machine with the longest (or short-

Algorithm 1 Local search heuristic

given a solution $\mathbf{x} = (x_1, \ldots, x_n)$ with makespan $f(\mathbf{x})$
$\mathbf{x}^* \leftarrow \mathbf{x}$ and $f(\mathbf{x}^*) \leftarrow f(\mathbf{x})$
repeat
  $\mathbf{x}' \leftarrow \mathbf{x}^*$
  $c \leftarrow max\_machine(\mathbf{x})$
  $d \leftarrow min\_machine(\mathbf{x})$
  $j \leftarrow min\_job(\mathbf{x}, d, c)$
  $x_j \leftarrow d$
  if $f(\mathbf{x}') < f(\mathbf{x})$ **then**
    $\mathbf{x}^* \leftarrow \mathbf{x}'$ and $f(\mathbf{x}^*) \leftarrow f(\mathbf{x}')$
  end if
until no improvement is made
$\mathbf{x} \leftarrow \mathbf{x}^*$ and $f(\mathbf{x}) \leftarrow f(\mathbf{x}^8)$

est) completion time from solution $\mathbf{x}$. The $min\_job(\mathbf{x}, d, c)$ function gives the job currently assigned to machine c with the shortest processing time from machine d. With a given solution $\mathbf{x}$, the heuristic will first identify the machine c with the longest completion time (equal to the makespan). Then, the job j with the shortest processing time on machine c from machine c will be removed and inserted into the machine d with the shortest completion time to generate the new solution $\mathbf{x}'$. An example of this heuristic is shown in Figure 3. In this case, job 13 from machine 1 is moved to machine 2. If this move improves the overall makespan of solution then the change is accepted ($\mathbf{x}^* \leftarrow \mathbf{x}'$) and the process is repeated until no improvement is made. The key principle of this heuristic is to balance the workload of machines in order to reduce the makespan. Because only a simple change is made in each iteration, the new solution $\mathbf{x}$ can be reevaluated very quickly (by subtracting the processing time p_{jc} from C_c and adding p_{jd} from C_d and finding C_{max}).

D. The overall algorithm

Algorithm 2 shows the pseudo code for the proposed PSO algorithm. $pbest^{\mathbf{x}}$ is the personal best position found by particle $\mathbf{x}$ and $gbest$ is the best position found by the swarm P. The $rand()$ function is used to generate the random number from 0 to 1 and $rand\_int(m)$ is used to generate random integer number from 1 to m.

The algorithm starts by randomly generating the initial solutions (simply by assigning a job to a random machine). In each iteration of the proposed algorithm, all particles will be evaluated to find the makespans as well as update $pbest^{\mathbf{x}}$ and $gbest$. Then, the updating mechanism as described in Section II-B is applied to update the position of each particle $\mathbf{x}$ in the swarm P. Finally, the local search heuristic is applied to the new particle' position to improve the makespan if a predefined condition is met. In this paper, we will investigate two versions of this algorithm. The pure algorithm without local search is referred to as NDPSO and the hybrid one with the local search heuristic is referred to as HDPSO.

Algorithm 2 Proposed algorithm

initialize positions of particles in the swarm P randomly
repeat
  for each particle $\mathbf{x} \in P$ **do**
    $f(\mathbf{x}) \leftarrow$ find the makespan of $\mathbf{x}_k$
    if $f(\mathbf{x}) < f(pbest^{\mathbf{x}})$ **then**
      $pbest^{\mathbf{x}} \leftarrow \mathbf{x}$
    end if
    if $pbest^{\mathbf{x}} < gbest$ **then**
      $gbest \leftarrow pbest^{\mathbf{x}}$
    end if
  end for
  for each particle $\mathbf{x} \in P$ **do**
    for $j = 1 \ldots n$ **do**
      if $rand() < u$ **then**
        if $rand() < p$ **then**
          $x_j \leftarrow pbest^{\mathbf{x}}_j$
        else
          $x_j \leftarrow gbest_j$
        end if
      end if
      if $rand() < m$ **then**
        $x_j \leftarrow rand\_int(m)$
      end if
    end for
    if condition is met **then**
      apply local search in Algorithm 1 to $\mathbf{x}$
    end if
  end for
until terminating criteria

III. Experimental design

This section shows the parameters settings of NDPSO, HDPSO, and existing PSO methods in our experiments. Then, the datasets used to evaluate the performance of our proposed algorithm will be described.

A. Parameter settings

To evaluate the effectiveness of the proposed methods, we will compare them against two PSO methods for GCS in the literature, CPSO [1] and DPSO [26]. For each PSO method, 50 independent runs are made for a problem instance. Based on the recommendations from their respective papers, DPSO uses self and social recognition components of 2.0, velocities from -40 to 40. Meanwhile, CPSO algorithm used a starting weight of 0.9 which linearly decreases to 0.1 through generations, self and social recognition components of 1.49 and position, and velocities from -1.0 to 1.0.

Pilot experiments of our proposed methods suggests that these methods work well with a *u* value starting at 0.7 which linearly decreases to 0.3, a *p* value of 0.3 and a mutation rate of 0.005. For HDPSO, the local search heuristic is applied to all particles every ten generations. This is done to reduce the computational costs that may caused by the local search

heuristic. All four algorithms have 100 particles and were run for 300 generations on each problem instance. In the initial population, one solution will be generated by using the min-min heuristic [26], [2], which has been shown to be very effective to reduce the makespan. Other initial solutions will be randomly generated.

B. Datasets

All four algorithms were tested on two grid scheduling datasets, one from [8] and the second from [1]. The first dataset contains 1200 problem instances, divided into twelve equal categories. Each category is defined by three characteristics, *consistency* (consistent, inconsistent, and partially-consistent), *job heterogeneity* (high or low) and *machine heterogeneity* (high or low). Consistency refers to whether the ranking of fastest to slowest jobs on each machine follows the same ordering: consistent means the order is maintained, inconsistent means that it is not and partially-consistent means that there is a consistently ordered subset of jobs. Heterogeneity of machines and jobs refers to how similar the speed or length of the machines and jobs are, respectively: low means they are close together, high means they are spread out. All of the problem instances have 512 jobs and 16 machines. The second problem set has 21 consistent instances that vary in the number of jobs and machines. There are seven categories characterised by the number of jobs and the number of machines, each having three instances.

IV. Results

This section presents the results of the two proposed PSO methods and other PSO methods in the literature. Then, we examine the behaviours of our proposed PSO methods to understand its effectiveness.

A. Computational comparisons

To compare the quality of solutions obtained by different PSO methods for a problem instance, we use the relative deviation $(\%) = 100 \times (obj - obj^*)/obj^*$ where obj is the makespan of an instance obtained by a PSO method in one specific run and obj^* is the best solution of that instance found by all four methods (through all 50 independent runs). In our experiments, the better methods are the ones with lower relative deviations. Table I and Table II showed the minimum, average, maximum and standard deviation (Std.) of relative deviations obtained by each PSO method for 100 instances and 50 independent runs in each subset (12 subsets for Braun et al. [8] dataset and 7 subsets for Liu et al. [1] dataset). The "Time" column shows the average running time of the considered PSO method for an instance in the subset. In Table I, u-x-y-z is used to represent the characteristics of a subset in Braun et al. [8] dataset where x is the consistency, y is the job heterogeneity, and z is the machine heterogeneity.

In general, CPSO shows the worst performance among the four proposed PSO methods. One of the reasons is that the representation in CPSO cannot directly represent the GCS solutions. Therefore, its is very difficult for CPSO to utilise effectively useful information from personal best and global best particles to guide the search towards better solutions. DPSO is slightly better than CPSO in most cases but the improvement is not very obvious and it still produce poor results in some runs (see the "max" column).

NDPSO and HDPSO are the two most effective methods in our experiments. NDPSO are able to produce better solutions as compared to DPSO and CPSO in all subsets. The gaps between NDPSO and CPSO/DPSO are more obvious when dealing with inconsistent and partial-consistent problem instances. This indicates that the new updating mechanism in NDPSO can help it maintain a good diversity in the swarm while effectively exploring for good solutions, which is necessary to deal with tricky inconsistent or partial-consistent problem instances.

In overall, HDPSO provides the best results for all subsets in both datasets. The results in Table I and Table II show that HDPSO is significantly better than other PSO methods in all cases. For all subsets in [8] dataset, the average relative deviations of HDPSO are always below 4% and its maximum relative deviations are even better than the average relative deviations of other methods. This confirms both the robustness and the effectiveness of HDPSO. In Table II, the results show that HDPSO is relatively more effective than other methods as the size of the problem increases.

Z-tests (with $\alpha = 0.01$) of the results showed that makespan produced by both proposed algorithms is significantly shorter than that produced by the CPSO for most problem instances in both datasets. NDPSO outperforms DPSO for 1194 of the 1200 instances in the first dataset and 18 of the 21 instances in the second dataset. HDPSO are significantly better than other methods on all problems instances from the two datasets. These results suggest that HDPSO is a good method to cope with large-scale problem instances.

Regarding the computational times, CPSO is the slowest method. As discussed in Section I, the traditional updating mechanism in CPSO may be quite time consuming when dealing with large-scale instances. DPSO is more efficient than CPSO because it uses a more compact representation and a fast updating mechanism [26]. NDPSO is more efficient than DPSO and CPSO for all subsets. HDPSO is slightly slower than NDPSO because of the local search heuristics but it is still faster than DPSO and CPSO. As the size of the problem increases in Table II, HDPSO is still very efficient. This again indicates that HDPSO has good scalability, which is suitable to deal with large-scale problem instances.

B. Further analysis of proposed PSO methods

Figures 4–6 respectively shows the best/average/worst makespan found by the four PSO methods during their search for instances in the **u-c-hi-hi** subset (values in the figures are the average makespan from 50 independent runs and 100 instances obtained by each method).

In Figure 4 and Figure 5, it is easy to see that CPSO improves its solutions in a very low rate as compared to other PSO methods. Another interesting observation is that

Table I: Relative deviation (%) in makespan and running time (in seconds) for Braun et. al. [8] dataset

Subsets	NDPSO					HDPSO					DPSO					CPSO				
u-x-y-z	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time
u-c-hi-hi	1.54	4.29	9.21	1.22	1.39	0.00	0.73	2.34	0.37	1.87	2.14	5.70	18.73	1.51	3.01	2.14	5.99	12.60	1.44	11.02
u-c-hi-lo	0.38	1.63	4.13	0.46	1.39	0.00	0.28	1.14	0.16	1.97	0.92	2.52	5.80	0.65	3.11	0.92	2.71	5.70	0.67	11.14
u-c-lo-hi	1.45	4.34	8.92	1.12	1.38	0.00	0.74	2.05	0.37	1.84	1.97	5.70	11.51	1.27	3.02	1.81	5.87	12.43	1.25	11.01
u-c-lo-lo	0.50	1.64	3.84	0.42	1.39	0.00	0.28	0.97	0.15	1.93	0.84	2.49	5.04	0.62	3.11	1.03	2.75	5.55	0.69	11.14
u-i-hi-hi	1.70	7.61	23.11	2.45	1.42	0.00	0.99	3.62	0.51	2.16	3.24	11.86	29.67	3.70	3.25	3.75	12.04	28.45	3.57	11.38
u-i-hi-lo	0.69	3.15	8.31	0.93	1.42	0.00	0.61	2.27	0.31	2.03	1.39	4.85	11.11	1.32	3.25	1.39	5.19	10.89	1.36	11.39
u-i-lo-hi	1.19	7.47	24.12	2.82	1.42	0.00	1.00	3.61	0.52	2.13	2.37	11.37	33.60	4.13	3.25	2.11	11.49	32.81	3.94	11.41
u-i-lo-lo	0.73	3.17	8.21	0.98	1.42	0.00	0.59	2.00	0.30	1.99	1.35	4.78	10.75	1.29	3.25	1.77	5.08	12.13	1.33	11.39
u-p-hi-hi	0.26	5.44	17.04	2.07	1.41	0.00	1.11	3.82	0.58	2.00	0.96	8.32	26.44	2.82	3.17	1.07	8.52	26.44	2.76	11.25
u-p-hi-lo	0.67	2.54	7.05	0.78	1.41	0.00	0.49	1.75	0.27	1.97	0.97	3.83	8.70	1.05	3.20	1.30	4.15	9.69	1.11	11.29
u-p-lo-hi	0.80	5.30	14.05	1.85	1.41	0.00	1.11	3.73	0.59	1.97	0.91	7.95	21.77	2.56	3.17	1.71	8.14	24.63	2.51	11.24
u-p-lo-lo	0.59	2.52	7.22	0.78	1.41	0.00	0.49	1.70	0.26	1.94	1.35	3.84	8.49	1.04	3.20	1.51	4.10	9.27	1.08	11.29

Table II: Relative deviation (%) in makespan and running time (in seconds) for Liu et al. [1] dataset

Subsets	NDPSO					HDPSO					DPSO					CPSO				
$m \times n$	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time	Min	Avg.	Max	Std.	Time
3×13	0.00	0.55	2.65	0.48	0.09	0.00	0.08	0.59	0.12	0.09	0.00	0.49	1.94	0.42	0.10	0.00	1.09	3.56	0.78	0.22
5×100	0.03	0.15	0.41	0.07	0.22	0.00	0.01	0.03	0.01	0.28	0.15	0.63	1.51	0.23	0.37	0.16	0.73	1.47	0.22	1.30
8×60	0.12	1.10	3.70	0.60	0.13	0.00	0.04	0.15	0.03	0.17	0.92	3.31	6.40	1.06	0.25	1.12	4.44	8.20	1.49	0.89
10×50	0.87	3.41	12.19	2.38	0.11	0.00	0.16	0.59	0.11	0.14	2.25	8.73	16.25	3.17	0.23	5.80	12.27	16.25	2.94	0.82
10×100	0.22	0.83	3.26	0.44	0.22	0.00	0.06	0.26	0.04	0.28	1.35	4.04	8.60	1.32	0.45	1.94	4.62	8.60	1.19	1.59
60×500	5.58	7.76	9.81	1.30	1.22	0.00	0.46	0.99	0.21	2.01	8.41	10.42	12.23	1.18	6.68	8.63	10.83	12.23	1.17	28.18
100×1000	7.73	9.01	10.18	0.53	2.66	0.00	0.57	1.17	0.23	4.42	7.99	9.60	10.50	0.48	20.43	8.99	9.70	10.50	0.52	108.06

the average makespans of particles in CPSO improves in the early generations (moving down as shown in Figure 5) and dramatically get worse (moving up) before they gradually improve again. The reason for this behaviour is that the solution found by the min-min heuristic is quite good and particles in CPSO tends to improve its makespan as they converge towards the min-min solution. However, after a few early generations, CPSO is able to find better solutions and particles will be reguided to these new solutions. Because of the indirect representation of CPSO and its updating mechanism, the shift towards the new better solutions causes large, possibly negative (as shown in Figure 6), changes in the particles' positions (as well as the corresponding GCS solutions). When CPSO is trapped at a local optimum, no large change will be made and the swarm will converge to that solution.

DPSO can cope with the changes better than CPSO and the average makespans of DPSO gradually are reduced through generation. This is because DPSO uses a more appropriate updating mechanism based on its discrete representation. The disadvantage is that the changes DPSO makes for each updated particles is quite small and it will takes a while for DPSO to improve its solutions (see Figure 4).

The problems with CPSO and DPSO have been overcome

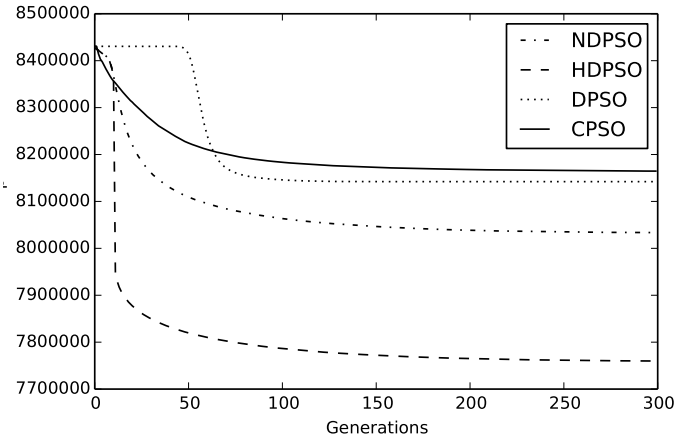

Figure 4: Best makespan obtained from PSO methods through generations for the **u-c-hi-hi** subset

with NDPSO. As shown in Figures 4–6, NDPSO can quickly improve the solutions, even better than those found by CPSO in the early generation, while maintaining a good stability in the swarm. This is a good evidence to confirm the effectiveness

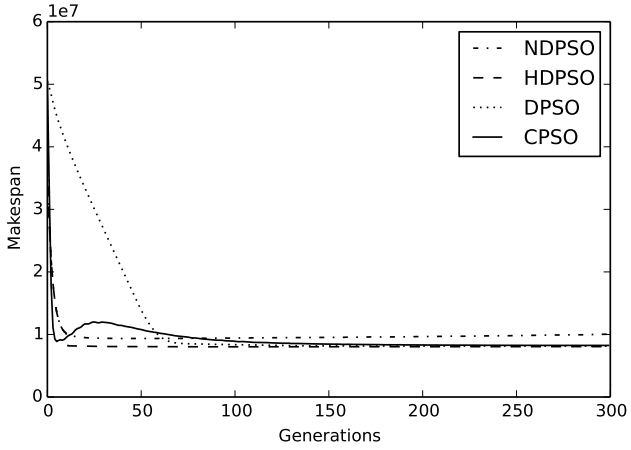

Figure 5: Average makespan obtained from PSO methods through generations for the **u-c-hi-hi** subset

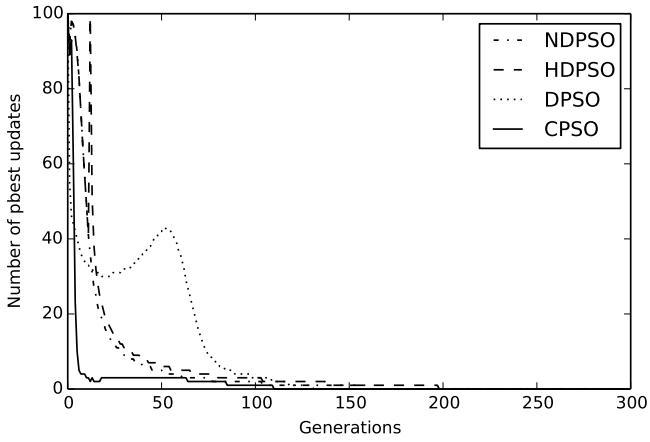

Figure 7: Number of personal best updates through generations for the **u-c-hi-hi** subset

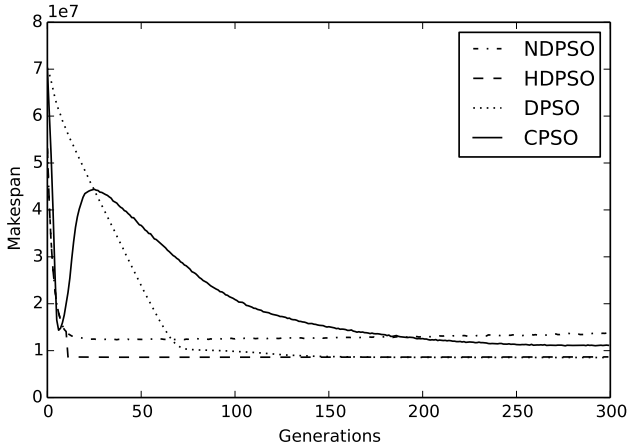

Figure 6: Worst makespan obtained from PSO methods through generations for the **u-c-hi-hi** subset

of the new proposed updating mechanism in NDPSO.

HDPSO is again shown to be the best methods here. With the support of the local search heuristic, HDPSO can dramatically improve solutions obtained by NDPSO, Within the first 20 generations, HDPSO has been able to find solutions that better than those obtained by other PSO methods from 300 generations. These results show that the local search heuristic plays a very important role in PSO in order to effectively and efficiently explore high quality solutions.

To gains more insights about the four PSO methods, we also look at the number of times personal best in updated, as shown in Figure 7. It is noted that the number of personal best updates for HDPSO has increased suddenly around generation 10 because that is when the local search heuristic is applied to refine the solutions found by the swarm. In general, this figure also support our previous explanation that NDPSO as well as HDPSO can overcome the dramatical changes in CPSO and the slow improving rate in DPSO. We can see that the number of personal best updates of NDPSO and HDPSO are somewhere between those of CPSO and DPSO. This suggests that NDPSO and HDPSO can balance quite well between exploration and explication.

Figure 8 shows total number of possible machine-to-job assignments (TNA) for all jobs from solutions in the swarm. This indicator is used to check how many machine-to-job assignments have been considered. The minimum value of TNA is the number of jobs n when solutions from all particles are the same. The maximum of TNA is $m * n$ when all possible machine-to-job assignments are presented in solutions of the swarm. This indicator is useful to measure the diversity of solutions in PSO when dealing with GCS. From Figure 8, it is easy to see that TNA of CPSO and DPSO have reduced through generations when they converge to global best solutions. On the other hand, NDPSO and HDPSO still maintain high TNA through generations. One of the the key reasons for this behaviour is the use of mutation operator at the end of each generation of NDPSO and HDPSO. This mutation operator will help produce new machine-to-job assignments and allow NDPSO and HDPSO to explore new solutions even at the end of their search.

V. Conclusions

This paper has developed a new PSO method for GCS problems with independent jobs. The key idea of the new proposed method is to create a more direct approach to position updating in PSO, which can cope with the discrete characteristic of GCS solutions. The new updating mechanism developed in this paper makes the proposed PSO more effective and efficient by directly using the discrete solutions from personal and global best particles to update new positions. A new local search heuristic is also developed and used within the proposed PSO to refine the solutions in the swarm. The experimental results show that the proposed PSO and its hybrid version

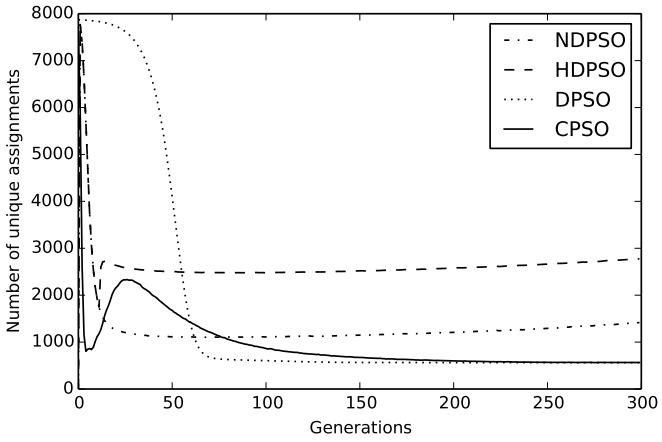

Figure 8: Number of possible machine-to-job assignments through generations for the **u-c-hi-hi** subset

are very effective and efficient as compared to other PSO methods proposed in the literature. In general, the hybrid method outperforms all other methods on all testing instances. It also shows great performance on large instances when other methods fail to provide good results. Regarding the computational times, the hybrid method is still much faster than other PSO methods in the literature.

The results and analyses have suggested that representation and updating mechanism play very important roles in PSO when dealing with discrete problems such as GCS. In the future studies, it would be interesting to investigate the performance of the new updating mechanism when it is applied to other scheduling and optimisation problems. Another important issue that need to be considered is to create a more adaptive version of the proposed PSO in order to help it cope better with a wide range of problems.

REFERENCES

[1] H. Liu, A. Abrahamc, and A. E. Hassanien, "Scheduling jobs on computational grids using a fuzzy particle swarm optimisation algorithm," *Future Generation Computer Systems*, vol. 26, pp. 1336–1343, 2010.

[2] F. Xhafa, J. Carretero, B. Dorronsoro, and E. Alba, "A tabu search algorithm for scheduling independent jobs in computational grids," *Computing and Informatics*, vol. 28, no. 2, pp. 237–250, 2009.

[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. Dong and S. G. Akl, "Technical report no. 2006-504 scheduling algorithms for grid computing: State of the art and open problems," Queen's University, Tech. Rep. 2006-504, 2006.

[5] E. G. Talbi and A. Y. Zomaya, Eds., *Grid Computing for Bioinformatics and Computational Biology*. John Wiley & Sons, 2007.

[6] Y. Lin, M. Pfund, and J. Fowler, "Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems," *Computers & Operations Research*, vol. 38, no. 6, pp. 901–916, Jun 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2010.08.018

[7] L.-H. Su and C.-Y. Lien, "Scheduling parallel machines with resource-dependent processing times," *International Journal of Production Economics*, vol. 117, no. 2, pp. 256–266, Feb 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ijpe.2008.10.014

[8] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, 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, 2001.

[9] A. Page and T. Naughton, "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms," *Artificial Intelligence Review*, vol. 24, no. 3-4, pp. 415–429, 2005.

[10] G. Ritchie and J. Levine, "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments," in *PLANSIG'04: Workshop of the UK Planning and Scheduling Special Interest Group*, 2004.

[11] ——, "A fast, effective local search for scheduling independent jobs in heterogeneous computing environments," in *PLANSIG'03: Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 2003, p. 178183.

[12] F. Xhafa, E. Alba, B. Dorronsoro, and B. Duran, "Efficient batch job scheduling in grids using cellular memetic algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 7, no. 2, pp. 217–236, 2008.

[13] A. YarKhan and J. Dongarra, "Experiments with scheduling using simulated annealing in a grid environment," in *Grid Computing - GRID 2002*, 2002, vol. 2536, pp. 232–242.

[14] F. Xhafa, B. Duran, A. Abraham, and K. Dahal, "Tuning struggle strategy in genetic algorithms for scheduling in computational grids," in *CISIM '08: Computer Information Systems and Industrial Management Applications*, 2008, pp. 275–280.

[15] H. Izakian and A. Abraham, "Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments," *Neural Network World*, vol. 19, no. 6, pp. 695–710, 2009.

[16] 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.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[18] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[19] D. Sha and C. Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, no. 4, pp. 791–808, 2006.

[20] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 400–407.

[21] M. Tasgetiren, M. Sevkli, Y.-C. Liang, and G. Gencyilmaz, "Particle swarm optimization algorithm for permutation flowshop sequencing problem," in *Ant Colony Optimization and Swarm Intelligence*. Springer, 2004, vol. 3172, pp. 382–389.

[22] C. Goh, K. Tan, D. Liu, and S. Chiam, "A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design," *European Journal of Operational Research*, vol. 202, no. 1, pp. 42–54, 2010.

[23] B. Xue, M. Zhang, and W. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.

[24] R. Perez and K. Behdinan, "Particle swarm approach for structural design optimization," *Computers & Structures*, vol. 85, no. 1920, pp. 1579 – 1588, 2007.

[25] W.-j. Xia and Z.-m. Wu, "A hybrid particle swarm optimization approach for the job-shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 29, no. 3-4, pp. 360–366, 2006.

[26] H. Izakian, B. T. Ladani, A. Abraham, and V. Snasel, "A discrete particle swarm optimization approach for grid job scheduling," *International Journal of Innovative Computing, Information and Control*, vol. 6, no. 9, 2010.