Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Novel Discrete Differential Evolution Algoritnm for Task Scheduling in Heterogeneous Computing Systems

Qinma Kang*
School of Electronics and Information Engineering
Tongji University
Shanghai 201804, China

Hong He
School of Information Engineering
Shandong University at Weihai
Weihai 264209, China

***Abstract*—Task scheduling is one of the core steps to effectively exploit the capabilities of distributed heterogeneous computing systems. In this paper, a novel discrete differential evolution (DDE) algorithm is presented to address the task scheduling problem. The encoding schemes and the adaptation of classical differential evolution algorithm for dealing with discrete variables are discussed as well as the technique needed to handle boundary constraints. The performance of the proposed DDE algorithm is showed by comparing it with a genetic algorithm, which is a well-known population-based probabilistic heuristic, on a large number of randomly generated instances. Experimental results indicate that the proposed DDE algorithm has generated better results than GA in terms of both solution quality and computational time.**

***Keywords*—Discrete differential evolution algorithm; Heterogeneous computing; Task scheduling; Genetic algorithm**

I. INTRODUCTION

Heterogeneous computing systems (HCS) are composed of a heterogeneous suite of machines with varied computational capabilities interconnected by high-speed networks, and have emerged as a powerful platform to execute computationally intensive applications that have diverse computational requirements [1]. One of the major challenges for harnessing the computing power of HCS to achieve high performance for tasks is the scheduling problem, i.e., we need address that each task should be mapped to its suited machine architecture. The mapping methods can be grouped into two categories: on-line mode and batch-mode [2]. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks that is considered for mapping at the mapping events is called a meta-task [1, 3]. In this study, we consider the meta-task scheduling problem, which is to find a task assignment scheme that minimizes the schedule length of a meta-task .

The meta-task scheduling problem is known to be NP-complete [2]. Therefore, only small-sized instances of the scheduling problem can be solved optimally within a reasonable computational time using exact algorithms. Heuristic algorithms, on the other hand, have generally acceptable computation time and memory requirements to obtain a near-optimal or optimal solution. For this reason, most research focused on developing heuristic algorithms. Some of the well-known fast constructive heuristics, which have been reported in the literature, include Min-min, Max-min, Sufferage [3], Qos Guided Min-min [4], Segmented Min-min [5], etc. In recent years, a growing body of literature suggests the use of meta-heuristic search methods for combinatorial optimization problems (COPs). Several meta-heuristic algorithms that have been identified as having great potential to address practical optimization problems include Genetic Algorithms (GA) [6, 7], Simulated Annealing (SA), Tabu Search (TS) and Particle Swarm Optimization (PSO), etc. Consequently, over the past few years, several researchers have demonstrated the applicability of these methods to task assignment problem [8-11]. A good overview of the task assignment algorithms can be found in [12]. Due to the intractable nature of the matching and scheduling problem and its importance in HCS, new efficient techniques are always desirable to obtain the best possible solution within an acceptable computational time.

Differential evolution (DE) is a population-based meta-heuristic algorithm recently proposed by Storn and Price [13] for global optimization over continuous spaces. In the mutation process of a DE algorithm, the weighted difference between two randomly selected population members is added to a third member to generate a mutated solution. Then, a crossover operator follows to combine the mutated solution with a target solution so as to generate a trial solution. Thereafter, a selection operator is applied to compare the fitness function value of both competing solutions, namely, target and trial solutions to determine who can survive for the next generation. Since its invention, DE has been applied with high success to many numerical optimization problems. Moreover, different variants of the basic DE model have been proposed to improve its efficiency over continues problems.

Despite the simplicity and the high efficiency of DE, its application on the solution of COPs with discrete decision

*The author is also affiliated to School of Information Engineering, Shandong University at Weihai, Weihai 264209, China.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

variables is still limited. The major obstacle of successfully applying a DE algorithm to solve COPs in the literature is due to its continuous nature. To remedy this drawback, this research introduces a novel discrete differential evolution (DDE) algorithm for solving meta-task scheduling problem to effectively exploit the capabilities of heterogeneous computing systems.

The remaining paper is organized as follows. Section 2 gives a brief introduction to DE. Section 3 describes the formulation of meta-task scheduling problem. Section 4 introduces the discrete differential evolution algorithm. Section 5 reports the comparative performance and convergence analysis. Finally, section 6 summarizes the concluding remarks.

II. Brief Introduction to DE

As with all evolutionary optimization algorithms, DE starts with a population of *D*-dimensional search-variable vectors called individuals, where *D* corresponds to the number of problem's parameters, and each vector represents potential solution for the optimization problem at hand. It finds the global optima by utilizing the distance and direction information according to the differentiations among population. Currently, there are several variants of DE, which vary based on the base vector to be perturbed, the number and selection of the difference vectors and the type of crossover operators. The common format of DE is *DE/x/y/z*, where x represents a base vector to be perturbed, y is the number of difference vectors used for perturbation of x, and z stands for the type of crossover being used (bin: binomial; exp: exponential). Let t denote the number of iterations, the *i*-th vector of the population at the current generation can be represented by $X_i^t = (X_{i,1}^t, X_{i,2}^t, ..., X_{i,D}^t)$. Let *PS* denote the number of individuals in the initial population, namely population size, and the population size is usually kept constant throughout iterations. For a minimization problem, the basic procedure of DE, which is denoted as *DE/rand/1/bin* [13], can be given below:

Step 1: Initialize population.

For each search-variable, there may be a certain range within which value of the parameter should lie for better search results. At the very beginning of a DE run or at $t = 0$, problem parameters or independent variables are initialized somewhere in their feasible numerical range. Therefore, if the *j*-th parameter of the given problem has its lower and upper bound as x_j^L and x_j^U respectively, then we may initialize the *j*-th component of the *i*-th population members as

$$X_{i,j}^0 = x_j^L + rand(0,1) \times (x_j^U - x_j^L) \tag{1}$$

where rand(0,1) represents a uniformly distributed random value that ranges from 0 to 1.

Step 2: Mutation phase.

In the mutation phase, for each target vector X_i^t , $i \in \{1, 2, \ldots, PS\}$, a mutant vector V_i^t is obtained by

$$V_i^t = X_{r_1}^t + F \times (X_{r_2}^t - X_{r_3}^t) \tag{2}$$

where $r_1, r_2, r_3 \in \{1, 2, \ldots, PS\}$ are mutually distinct random indices and are also different from the current target index *i*. The vector $X_{r_1}^t$ is known as the base vector to be perturbed, and $F > 0$ is a scaling parameter.

Step 3: Crossover phase.

For each target X_i^t , a trial vector Y_i^t is formed as:

$$Y_{i,j}^t = \begin{cases} V_{i,j}^t & if \ \ rand(0,1) < CR \ \ or \ \ j = k \\ X_{i,j}^t & else \end{cases} \tag{3}$$

where *k* is a randomly chosen integer in the set {1, 2, ..., *D*}; the subscript *j* represents the *j*-th component of respective vectors; rand(0,1)$\in$(0, 1), drawn randomly for each *j*; *CR*$\in$(0, 1) is crossover probability that affects the convergence rate and robustness of the search process.

Step 4: Selection phase.

The selection scheme of DE also differs from the other evolutionary algorithms. In the selection phase, the function value of the trial vector, $f(Y_i^{t+1})$, is compared to $f(X_i^t)$, to determine which one of the target vector and the trial vector will survive in the next generation. The process may be outlined as:

$$X_i^{t+1} = \begin{cases} Y_i^t & if \ \ f(Y_i^t) < f(X_i^t) \\ X_i^t & else \end{cases} \tag{4}$$

where f is the function to be minimized. So if the new trial vector yields a better value of the fitness function, it replaces its target in the next generation; otherwise the target vector is retained in the population. Hence the population either gets better or remains constant but never deteriorates.

Recombination (mutation and crossover) and selection continues until a stopping criterion is satisfied, and then the best individual of the population found so far is reported as the final solution.

The scheme described above is only one variant of the basic DE algorithm. There are some other DE variants, mainly differing in the way they create the mutant vector (Equation (2)). The creation of the *i*-th mutant vector in another well-known variant called *DE/best/1/bin* [13] is given by the following relation:

$$V_i^t = X_{best}^t + F \times (X_{r_1}^t - X_{r_2}^t) \tag{5}$$

where X_{best}^t is the best vector of the current population.

III. Meta-task Scheduling Problem

A meta-task is defined as a collection of independent tasks that is considered for mapping at the scheduling events, and makespan the completion time for the entire meta-task [12]. A typical example of meta-task scheduling is the mapping of an arbitrary set of independent tasks from different users waiting to execute on a heterogeneous suite of machines. Each task in a meta-task may have associated properties, such as a deadline or a priority. It is assumed that each machine executes a single task at a time, in the order in which the tasks arrived, and the

size of the meta-task (number of tasks to execute), N, and the number of machines in the HCS, M, are static and known a priori. Therefore, task scheduling problem can be defined as N tasks are assigned on M heterogeneous resources with the objective of minimizing the completion time and utilizing the resources effectively.

To formulate the problem, define $P = \{p_1, p_2, \ldots, p_M\}$) as the set of M heterogeneous machines and $T = \{t_1, t_2, \ldots, t_N\}$ as the set of N independent tasks to be assigned to the machines. Because of the heterogeneous nature of the machines and dissimilar nature of the tasks, the expected execution times of a task on different machines are different. Every task has an expected time to compute (*ETC*) on a specific machine. We assume that the ETC of each task on each machine is known based on user-supplied information, task profiling and analytical benchmarking. The assumption of such *ETC* information is a common practice in scheduling research (e.g. [2-5]). So we can obtain an $N \times M$ *ETC* matrix. *ETC* (i, j) is the estimated execution time for task i on machine j. The total execution time used by machine p_j is the sum of the expected execution times of those tasks that are mapped to the machine. The makespan of a schedule is the maximum of the total execution times of those machines. To calculate the makespan of a schedule, we first introduce a vector of machine completion time which size is M [14]. Formally, for a machine p_m and a schedule S, the completion time of p_m is defined as follows:

$$completion[m] = \sum_{S[j]=m, \forall t_j \in T} ETC(j,m) \qquad (6)$$

where $S[j]=m$ denotes the task t_j is assigned to machine m under the schedule S.

We can use the completion time of machines to compute the makespan as follows:

$$makespan = \max\{completion[m] \mid \forall p_m \in P\} \qquad (7)$$

The objective of this work is to find a schedule scheme with the minimum makespan.

IV. The Proposed DDE Algorithm

A. Solution Representation and Initial Swarm Generation

One of the key issues in designing a successful DE algorithm is the solution representation, i.e. finding a suitable mapping between problem solution and DE individual. The natural coding for task scheduling problems is the integer vectors as used in GAs [12]. In this paper, each individual S corresponding to a feasible solution of the underlying problem is represented as a vector of size N in which its j-th position (an integer value) indicates the machine to which task j is assigned: $S[j] = m$, $m \in \{1, 2, \ldots, M\}$.

The DE is a population-based searching paradigm that explores the solution space by recombining a number of individuals, called a population. An initial population is randomly generated for iterative improvement according to equation (8), which is presented for the j-th dimension of the i-th particle.

$$X_{i,j}^0 = INT(1 + rand(0,1) \times M) \quad 1 \le i \le PS, 1 \le j \le N \qquad (8)$$

where INT is a function for converting a real-value to an integer value by truncation and rand(0,1) is a uniform random number in the range (0,1).

B. Fitness Value

In DE algorithm, all individuals at each iteration step are evaluated according to a measure of solution quality, called fitness. For the problem at hand, we use the makespan of an individual as its fitness value. See section 2.

C. The Proposed DDE Algorithm

In its canonical form, the DE algorithm is only capable of handling continuous variables. In order to apply the DE algorithm to integer optimization problems, it is crucial to design a suitable encoding scheme that maps the floating-point vectors to integer vector solutions. Several approaches have been used to deal with discrete variable optimization. Most of them round off the variable to the nearest available value before evaluating each trial vector, i.e., integer values are used to evaluate the objective function, even though DE itself may still works internally with continuous floating-point values [15]. Another widely used representation technique that deals with floating-point vectors for discrete optimization problems is the random-keys encoding. According to this technique, each solution is encoded as a vector of N floating-point numbers. The components of the vector are sorted and their order in the vector determines the final solution to the problem. However, this encoding scheme has been demonstrated to be poor suited within the DE algorithm to address discrete optimization problems [16]. Moreover, the techniques used to deal with sequencing optimization problem cannot be applied directly to task scheduling in HCS generally.

It is important to note that DE is self-adaptive. At the beginning of the evolution process, the mutation operator of DE favors exploration since parent individuals are far away to each other. As the evolutionary process proceeds to the final stage, the mutation operator favors exploitation in that the population converges to a small region. As a result, the adaptive search step enables the evolution algorithm to perform global search with a large search step at the beginning and refine the population with a small search step at the end. Therefore, extending DE to optimization of integer variables is rather easy. Only the scaling parameter is set to 1. Based on this idea, we propose a discrete variant of DE denoted as *DE/i/1/bin*. In the proposed algorithm, a mutant vector V_i^t is obtained by

$$V_i^t = X_i^t + X_{r_1}^t - X_{r_2}^t \qquad (9)$$

where $r_1, r_2 \in \{1, 2, \ldots, PS\} \leftarrow$are mutually distinct random indices and are also different from the current target index i. The trial vector Y_i^t is formed as:

$$Y_{i,j}^t = \begin{cases} V_{i,j}^t & if \ \ rand(0,1) < CR \\ X_{best,j}^t & else \end{cases} \qquad (10)$$

Note that we use the i-th vector as base vector for the i-th target individual, and then the mutant vector is combined with the best individual of the current population for yielding the trial vector at each iteration step.

D. Boundary Constraints

It is important to notice that the recombination operation of DE is able to extend the search outside of the initialized range of the search space, which leads to illegal solutions. For the problem under investigation, it is essential to ensure that parameter values lie inside their allowed ranges after recombination. A simple way to guarantee this is to replace parameter values that violate boundary constraints with random values generated within the feasible range:

$$V_{i,j}^{t} = \begin{cases} INT(1 + rand(0,1) \times M / 2) & if\ V_{i,j}^{t} < 1 \\ INT(M / 2 + rand(0,1) \times (M / 2 + 1)) & if\ V_{i,j}^{t} > M \end{cases} \quad (11)$$

E. The DDE Algorithm

The details of the proposed DDE algorithm are presented in Fig. 1. The algorithm starts with an initial population of *PS* individuals. Each individual vector corresponds to a candidate solution under investigation. Then, all of the individuals are iteratively improved until one of stopping criteria is satisfied. When the algorithm is terminated, the incumbent global best individual and the corresponding fitness value are output as the optimal task scheduling scheme and the minimum makespan.

```
DDE
{   Initialize parameters.
    Generate PS individuals at random using equation (8).
    Evaluate the individuals and determine the best
        individual.
    Repeat
    {   Mutation step:
            Generate mutant individuals using equation (9);
            Test boundary constraints for the mutant
                individuals. If violation occurs, the value is
                dragged to the feasible range using
                equation (11).
        Crossover step:
            Generate trial individuals using equation (10).
        Selection step:
            Determine the individuals of the next
                generation using equation (4);
            Determine the best individual of the population.
    } until one of the stopping criteria is satisfied;
    Return the best individual and its fitness value.
}
```

Figure 1. Pseude Code of DDE Algorithm.

V. Experiment Results

To evaluate the efficiency and effectiveness of the proposed algorithm, intensive experiments have been conducted. A large simulation dataset is created which features different mapping situations. The proposed algorithm is compared with the genetic algorithm (GA) proposed by Braun et al. [12] because both of them are population-based evolutionary algorithms. Both the algorithms for meta-task scheduling are coded in MATLAB6.5 and experiments are executed on a Pentium Dual 1.6GHz processor with 1GB main memory running under Windows XP environment.

The GA is implemented with roulette wheel selection, one-point crossover operator, a mutation operator and an elitism mechanism for the comparison purpose. The parameters of GA are set as follows: crossover probability = 0.8, mutation probability = 0.1, population size = 100 and their GA finalizes when either 1000 iterations have been executed, or, when the chromosome elite have not varied during 150 iterations.

The crossover rate CR in DDE is determined experimentally by incrementing from 0.1 to 0.9 in 0.1 steps. In the simulation trials tested, the performance peak is detected with CR equal to 0.4. The size of the initial population is set to 100 and we use the same stopping criteria as that used in GA for the fair comparison purpose.

The simulation model presented in [12] is also employed here for our comparison study. In this model, characteristics of the *ETC* matrices are varied in an attempt to simulate various possible heterogeneous computing environments. This kind of variation is implemented by setting the value scope of random numbers used to produce *ETC* matrices. Firstly, an $N \times 1$ baseline column vector B is generated, where each element $B(i)$, $1 \leqslant i \leqslant N$, is a uniform random number between 1 and Φb. Then, the rows of the *ETC* matrix are constructed. Each element $ETC(i, j)$, $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M$, of *ETC* matrix is created by taking baseline value, $B(i)$, and multiplying it by a uniform random number $x(i, j)$, whose value is between 1 and Φr. Therefore, any given value in the *ETC* matrix is within the range $[1, \Phi b \times \Phi r]$.

To denote various kinds of mapping situations, the matrices are classified into 12 different types according to three metrics: task heterogeneity, machine heterogeneity and consistency. The task heterogeneity describes the amount of variance among the execution times of the tasks in the meta-task for a given machine, two possible values are defined: high and low. Machine heterogeneity describes the possible variation of the running time of a particular task across all the machines, and again has two values: high and low. Each of the different task and machine heterogeneities is modeled by using different Φb and Φr values: high task heterogeneity is represented by setting Φb = 3000 and low task heterogeneity is modeled using Φb = 100. High machine heterogeneity is represented by setting Φr = 1000, and low machine heterogeneity is modeled using Φr = 10.

An *ETC* matrix is considered consistent when, if a machine m_i executes task t_k faster than machine m_j, then m_i executes all the tasks faster than m_j. Inconsistency means that a machine is faster for some tasks and slower for some others. An *ETC* matrix is considered semi-consistent if it contains a consistent sub-matrix. All instances consist of 256 tasks and 16 machines and are labeled *u_x_yyzz* whose meaning is the following:

- *u* means uniform distribution (used in generating the matrices).
- *x* means the type of inconsistency (*c*–consistent, *i*–inconsistent and *s* means semi-consistent).

TABLE I. COMPARATIVE RESULTS FOR DDE AND GA

Instrance	GA			DDE			Improvment over GA	NRT
	Mavg(s)	Mstd	Tavg(s)	Mavg(s)	Mstd	Tavg(s)		
u_i_hihi	8.833*106	7.13	64.39	6.292*106	5.35	62.19	28.77%	1.03
u_i_hilo	1.175*105	0.72	61.92	0.940*105	0.54	59.03	20.00%	1.05
u_i_lohi	2.998*105	2.06	61.60	2.127*105	1.63	59.14	29.05%	1.04
u_i_lolo	4.053*103	66.5	62.53	3.075*103	72.4	61.39	24.13%	1.02
u_c_hihi	8.289*106	4.73	64.59	6.438*106	4.52	63.30	22.33%	1.02
u_c_hilo	1.135*105	0.35	61.73	1.021*105	0.29	59.54	10.04%	1.04
u_c_lohi	2.927*105	1.52	61.59	2.254*105	1.14	58.66	22.99%	1.05
u_c_lolo	3.871*103	79.5	62.32	3.377*103	58.2	61.31	12.76%	1.02
u_s_hihi	9.392*106	5.20	64.53	7.303*106	3.22	62.57	22.24%	1.03
u_s_hilo	1.192*105	0.60	61.77	1.049*105	0.47	59.18	11.99%	1.04
u_s_lohi	3.077*105	1.45	61.69	2.458*105	1.44	59.11	20.12%	1.04
u_s_lolo	4.061*103	73.8	62.43	3.446*103	11.1	61.44	15.14%	1.02
Average							19.96%	1.03

●*yy* indicates the heterogeneity of the tasks (*hi*–high, and *lo*–low).

●*zz* indicates the heterogeneity of the resources (*hi*–high, and *lo*–low).

This set of instances has been actually considered as the most difficult one for the scheduling problem in heterogeneous environments and it is the main reference in the literature.

The size of the initial population is set to 100 and we use the same stopping criteria as that used in GA for the fair comparison purpose. As performance metrics, we consider the quality of the solution (makespan) and the amount of CPU time used for the benchmarks. The percentage improvement is computed as:

$$\left(1-\frac{makespan_{DDE}}{makespan_{GA}}\right)\times 100\% \qquad (13)$$

and the normalized running time (*NRT*) is computed by dividing the total GA algorithm running time with DDE algorithm running time for the benchmark under consideration. Moreover, both GA and DDE are stochastic-based algorithms and each independent run of the same algorithm on a particular testing instance may yield a different result, we thus calculate the average makespan and the average computation time over 20 independent runs of each algorithm for every problem instance. And the standard deviation of makespans is also computed for each instance.

All experimental results are summarized in table 1, where Mavg, Mstd and Tavg, respectively, denote the average makespan, the standard deviation of makespans and the average computation time. It can be observed that the proposed DDE algorithm outperforms the GA in terms of both solution quality and computational time for all the instances. Improvement of DDE over GA is between 10.04% and 29.05% with an average of 19.96%. Moreover, GA on the average runs 1.03 times slower than DDE technique. The DDE is also superior in terms of standard deviation. These results indicate that the proposed DDE algorithm is a viable alternative for solving the task scheduling problem.

These results are not compared with the traditional methods since earlier studies of Braun et al. [12] show that GA based algorithm for meta-task assignment problems outperform other traditional approaches such as SA and TS. Also, our preliminary tests indicate that PSO based algorithm [11] proposed for homogeneous systems is poorly suited for medium to large-sized problems of meta-task scheduling problems in HCS.

To analyze the convergence behavior of the proposed algorithms, we have recorded the variation of the makespans of the two algorithms at each iteration step. Fig. 2 displays a typical run of two algorithms for solving a problem instance with the number of tasks being equal to 256 and that of machines 16. From the evolutional processes of the algorithms, we find that DDE delivers better performance than GA in terms of solution quality. The graph also indicates GA evolves slower than DDE after approximately 300 iteration steps and the decrement of makespan almost stagnates.

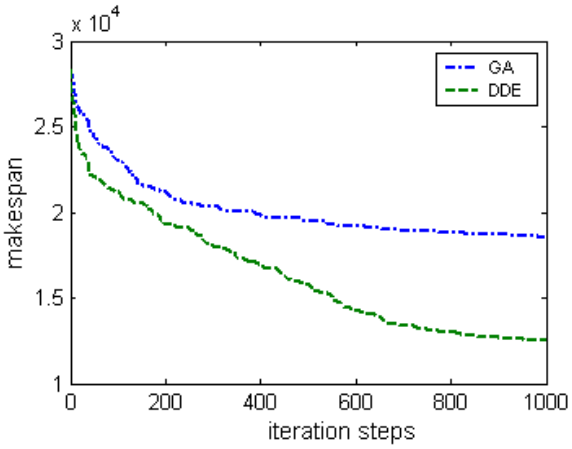

Figure 2. Convergence properties of GA and DDE.

VI. CONCLUSIONS

In distributed heterogeneous computing systems, qualified task scheduling among processors is an important step for efficient utilization of resources. In this paper, a new heuristic algorithm called DDE algorithm is proposed for the task scheduling problem in heterogeneous computing environments. DE is one of the recent evolutionary optimization methods. It

has been widely used in a wide range of applications. Unlike the standard DE, the DDE algorithm employs an integer-valued solution vector and work on the discrete domain. The performance of DDE algorithm is evaluated in comparison with a well-known GA algorithm for a number of randomly generated mapping problem instances. The results showed that the DDE algorithm solution quality is better than that of GA in all cases. Moreover, the DDE algorithm runs faster as compared with GA. These results indicate that the proposed DDE algorithm is an attractive alternative for solving the task scheduling problem. A natural extension to this work would be hybridizing DDE with a local search heuristic and a widespread search heuristic to promote a search for a global optimum. Moreover, DDE application to other combinatorial optimization problems needs further investigation.

REFERENCES

[1] H. J. Siegel and S. Ali, Techniques for mapping tasks to machines in heterogeneous computing systems. J. Syst. Architect., vol. 46, pp. 627–639, 2000.

[2] P. Luo, K. V. Lü and Zh. Zh. Shi, A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput., vol. 67, pp. 695-714, 2007.

[3] M. Maheswaran, S. Ali, et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In 8th IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, vol. 30-44, 1999.

[4] X. S. He, X. H. Sun and G. Laszewski, QoS guided min-min heuristic for grid task scheduling. J. Comput Sc. Technol., vol. 18, pp. 442-451, 2003.

[5] M. Y. Wu, W. Shu and H. Zhang, Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In Procceding of the 9th Heterogeneous Computing Workshop (HCW'00), Cancun, Mexico, pp. 375-385, 2000.

[6] D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, January 1989.

[7] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. The University of Michigan Press, Ann Arbor, 1975.

[8] R. Subrata, Y. A. Zomaya and B. Landfeldt, Artificial life techniques for load balancing in computational grids. J. Compu. Syst. Sci., vol. 73, pp. 1176–1190, 2007.

[9] G. Attiya and Y. Hamam, Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. J. Parallel Distrib. Comput., vol. 66, pp. 1259 – 1266, 2006.

[10] W. H. Chen and C. S. Lin, A hybrid heuristic to solve a task allocation problem. Comput. Oper. Res., vol. 27, pp. 287-303, 2000.

[11] A. Salman, I. Ahmad and S. Al-Madani, Particle swarm optimization for task assignment problem. Microprocess. Microsy., vol. 26, pp. 363-371, 2002.

[12] T. D. Braun, H. J. Siegel, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system. J. Parallel Distrib. Comput., vol. 6, pp. 810-837, 2001.

[13] R. Storn and K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim., vol. 11, pp. 241–354, 1997.

[14] J. Carretero and F. Xhafa, Using genetic algorithms for scheduling jobs in large scale grid applications. J. of Technological and Economic Development–A Research Journal of Vilnius Gediminas Technical University, vol. 12, pp.11–17, 2006.

[15] G. Onwubolu and Davendra D. Scheduling flow shops using differential evolution algorithm. Eur. J. Oper. Res., vol. 171, pp. 674–692, 2006.

[16] C. N. Andreas and L. O. Sotiris, Differential evolution for sequencing and scheduling optimization. J Heuristics, vol. 12, pp. 395–411, 2006.