

# Task scheduling in multiprocessor systems using inertial velocity differential evolution

**Xiaohong Qiu<sup>1\*</sup>, Yuting Hu<sup>2</sup>, Bo Li<sup>1</sup>**

<sup>1</sup>Software School, Jiangxi University of Science and Technology, Nanchang 330013, China

<sup>2</sup>School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China

Received 9 August 2014, www.cmmt.lv

---

## Abstract

Task scheduling in multiprocessor systems is a challenge NP-complete problem. All practical real-time scheduling algorithms in multiprocessor systems present a trade-off between their computational complexity and performance. In this paper, An improved Differential Evolution algorithm combined Particle Swarm Optimization idea is proposed to solve the Task Scheduling Problem (TSP) in multiprocessor system. The proposed algorithm called Inertial Velocity Differential Evolution (IVDE) consists of an additional inertial velocity factor based on adaptive differential evolution algorithm. IVDE optimizes task scheduling to the minimum of the overall schedule length. The simulation results show that IVDE algorithm not only reduces the computational complexity, but also is easy to get the global optimum compared with GA and Ant Colony Optimizer to solve the task scheduling problem in multiprocessor systems.

*Keywords:* multiprocessor systems, task scheduling problem, differential evolution; algorithm

---

## 1 Introduction

Many achievements in the field of computer technology in the last decades enabled the intensive use of multiprocessor systems. In these systems, the key to achieving high efficiency is the optimal scheduling of parallel programs on multiple processors. Multiprocessor scheduling involves the assignment of a set of partially ordered computational tasks onto a multiprocessor system such that the overall schedule length is minimized [1]. Task Scheduling Problem (TSP) belongs to the class NP-hard problems, even when the processors are fully connected and there are no communication delays [2]. There are many scientific papers in the area of scheduling tasks [3]. In most of these works, different scheduling algorithms evaluated on randomly generated graphs of business or operations on graphs that are modelled on actual applications [4]. A standard set of graph operations called. "Standard Task Graph (STG) Set" is introduced [4]. Commonly used graphs are not available to other researchers scheduling problem [5, 6]. Many task scheduling algorithms have been developed with moderate complexity as a constraint, which is a reasonable assumption for general purpose development platforms. Generally, the task scheduling algorithms may be divided in two main classes: greedy and non-greedy (iterative) algorithms [7].

Various researchers have proposed heuristic solutions to the task scheduling problem that capture the concurrent, iterative, and evolutionary nature characteristics of task execution [8, 9]. Recently there used an approach based

upon Generic Algorithms with different selection [9, 10], Ant Colony Optimization [11, 12]. Differential Evolution (DE) is a population-based random search heuristic parallel evolutionary algorithm. Improved DE such as JADE (adaptive differential evolution with optional external archive) [13] and CoDE (composite trial vector generation strategies differential evolution algorithm) [14] has achieved good successful application cases. But there are no attentions to apply them to solve the TSP. In this paper, we propose a new DE with additional inertial velocity factor to solve the TSP and test it with STG set.

The organization of the paper is as follows. After a general introduction of the Particle Swarm Optimization and Differential Evolution, an improved differential evolution called Inertial Velocity Differential Evolution (IVDE) with an additional inertial velocity factor is presented and discussed in section 2. Task scheduling problem is described in section 3. In section 4, a new state variable coder for IVDE and the individual fitness function is proposed to solve the TSP. IVDE optimizes the task scheduling to minimize the overall schedule length by differential evolution. IVDE compared with GA, Ant Colony Optimization on a set of graphs from the website below:

<http://www.kasahara.elec.waseda.ac.jp/schedule/index.html> [4] is discussed in section 5. The paper has been concluded in section 6.

---

\* *Corresponding author's* e-mail: [jxauqiu@163.com](mailto:jxauqiu@163.com)

**2 Improved inertial velocity differential evolution**

**2.1 CLASSIC PARTICLE SWARM OPTIMIZER (PSO) AND DIFFERENTIAL EVOLUTION (DE)**

We suppose that the minimized objective function is  $f(x_i)$ ,  $x_i = (x_{i,1}, \dots, x_{i,d}, \dots, x_{i,D}) \in R^D$ , the feasible solution space is  $\Omega = \prod_{i=1}^D [x_{i,\min}, x_{i,\max}]$ . And the initial

population  $P = \{x_1, \dots, x_i, \dots, x_{N_p}\}$  is randomly sampled from  $\Omega$ , where  $N_p$  is the population size. Particle Swarm Optimizer (PSO) is proposed to solve the optimal problem by Kennedy and Eberhart in 1995 [15]. The particle velocity and position of the formula is given by:

$$v_{i,d}^{k+1} = w_i^k v_{i,d}^k + c_1 r_1^k (x_{i,best,d}^k - x_{i,d}^k) + c_2 r_2^k (x_{gbest,d}^k - x_{i,d}^k), \quad (1)$$

$$x_{i,d}^{k+1} = x_{i,d}^k + v_{i,d}^k, \quad (2)$$

where the superscript  $k$  of a variable stands for the  $k$ -th iteration or  $k$ -th generation, the subscript  $i$  stands for the  $i$ -th particle, and the subscript  $d$  stands for the  $d$ -th sub dimension. Namely  $x_{i,d}^k, v_{i,d}^k$  is the position and speed in  $d$ -th sub dimension of the  $i$ -th particle in the  $k$ -th iteration(or generation);  $w_i^k$  is the inertia weight factor to avoid PSO into local optimum[15];  $c_1, c_2$  is the constant weight factors;  $r_1^k, r_2^k$  is the random number in the interval  $[0,1]$ ;  $x_{i,best,d}$  is currently the  $i$ -th particle individual best solution in its history;  $x_{gbest,d}$  is currently the global best solution in all particles' history.

Differential Evolution (DE) is always used to deal with the continuous optimization problem. At  $k$ -th generation, DE creates a mutant vector  $v_i = (v_{i,1}, \dots, v_{i,d}, \dots, v_{i,D}) \in R^D$  for each individual  $x_i$  in current population. Different mutation operation will play key role to decide the DE performance. One widely used DE mutation operator named *DE/current-to-best/1/bin* is shown as (3):

$$v_{i,d}^{k+1} = x_{i,d}^k + F_i^k \times ((x_{i,best,d}^k - x_{i,d}^k) + \lambda \times (x_{r1,d}^k - x_{r2,d}^k)), \quad (3)$$

where  $r1, r2$  are the distinct integers randomly selected from the range  $[1, N_p]$  and are also different from  $i$ . The parameter  $F_i$  is called the mutation factor, which amplifies the difference vectors.  $x_{i,best,d}$  is the  $d$ -th element of the best individual in the current population. After mutation, DE performs a binomial crossover operator on  $x_{i,d}$  and  $v_{i,d}$  to generate a trial vector  $u_{i,d}$ :

$$u_{i,d}^k = \begin{cases} v_{i,d}^k & \text{if } rand(1) \leq C_R \text{ or } d = rand(D) \\ x_{i,d}^k & \text{otherwise} \end{cases}, \quad (4)$$

where  $i=1,2,\dots,N_p, d=1,2,\dots,D$ ,  $rand(D)$  is a randomly chosen integer in  $[1,D]$ ,  $rand(1)$  is a uniformly distributed random number between 0 and 1 which is generated for each individual, and  $C_R \in [0,1]$  is called the crossover control parameter. Due to the use of  $rand(\cdot)$ , the trial vector  $u_i$  differs from its target vector  $x_i$ .

If the  $d$ -th element  $u_{i,d}$  of the trial vector  $u_i$  is infeasible (i.e., out of the boundary), it is reset as follows:

$$u_{i,d} = \begin{cases} \min\{x_{d,\max}, 2x_{d,\min} - u_{i,d}\} & \text{if } u_{i,d} < x_{d,\min} \\ \max\{x_{d,\min}, 2x_{d,\max} - u_{i,d}\} & \text{if } u_{i,d} > x_{d,\max} \end{cases}. \quad (5)$$

The selection operator is performed to select the better one from the target vector  $x_i^k$  ( $k$ -th generation) and the trial vector  $u_i^k$  to enter the next generation  $x_i^{k+1}$ :

$$x_i^{k+1} = \begin{cases} u_i^k & \text{if } f(u_i^k) < f(x_i^k) \\ x_i^k & \text{otherwise} \end{cases}. \quad (6)$$

**2.2 IMPROVED DE WITH ADDITIONAL INERTIAL VELOCITY FACTOR**

To combine the good feature of PSO with DE, we suggest an improved DE with additional Inertial Velocity Factor. After the inertial velocity factor added, the Equation (3) is rewritten as Equations (7) and (8).

$$vel_{i,d}^{k+1} = w_{i,d}^k \cdot vel_{i,d}^k + F_i^k \cdot ((x_{i,best,d}^k - x_{i,d}^k) + \lambda \cdot (x_{r1,d}^k - x_{r2,d}^k)), \quad (7)$$

$$v_{i,d}^{k+1} = x_{i,d}^k + vel_{i,d}^k, \quad (8)$$

where the superior  $k$  and the subscript  $i$  and  $d$  have the same meaning as (1). Namely  $vel_{i,d}^k$  is the speed in  $d$ -th sub dimension of the  $i$ -th particle in the  $k$ -th iteration (or generation),  $w_{i,d}^k$  is the inertia weight factor of velocity.  $F_i$  is the mutation factor, its value estimation method will use the same method used in JADE algorithm [13], and the “*DE/current-to-pbest/1*” strategy instead of only adopting the best individual in the “*current-to-best/1*” strategy. Namely  $x_{i,best,d}^p$  is randomly chosen as one of the top 100  $p\%$  individuals in the current population with  $p \in (0,1]$  instead of  $x_{i,best,d}$ . The Equation (7) will be:

$$vel_{i,d}^{k+1} = w_{i,d}^k \cdot vel_{i,d}^k + F_i^k \cdot ((x_{i,best,d}^p - x_{i,d}^k) + \lambda \cdot (x_{r1,d}^k - x_{r2,d}^k)), \quad (9)$$

where  $F_i^k$  is associated with  $x_i$  and is re-generated at each generation by the adaptation process introduced in (10). At  $k$ -th generation, the mutation factor  $F_i$  of each individual  $x_i$  is independently generated according to a Cauchy

distribution with location parameter  $\mu_F$  and scale parameter 0.1, and then truncated to be 1 if  $F_i \geq 1$  or regenerated if  $F_i \leq 0$ . Denote  $S_F$  as the set of all successful mutation factors in  $k$ -th generation. The location parameter  $\mu_F$  of the Cauchy distribution is initialized to be 0.5 and then updated at the end of each generation as (11), where  $c \in (0,1)$  is a positive constant and  $mean_L(\cdot)$  is the Lehmer mean calculated by (12).

$$F_i = randc(\mu_F, 0.1), \tag{10}$$

$$\mu_F = (1-c) \times \mu_F + c \times mean_L(S_F), \tag{11}$$

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}. \tag{12}$$

Similarly, we consider the crossover probability  $C_{R,i}$  also has the different weighting feature, and estimated by (13) and (14).

$$C_{R,i} = randn(\mu_{CR}, 0.1) + C_{R,iw}, \tag{13}$$

$$\mu_{CR} = (1-c) \times \mu_{CR} + c \times mean_A(S_{CR}), \tag{14}$$

where  $randn(\mu_{CR}, 0.1)$  is a normal distribution of mean  $\mu_{CR}$  and standard deviation 0.1,  $C_{R,iw}$  is a modified factor with different weighting factor according to their fitness. Denote  $S_{CR}$  as the set of all successful crossover probabilities  $C_{R,i}$ 's at  $k$ -th generation. The mean  $\mu_{CR}$  is initialized to be 0.5 and then updated at the end of each generation as (14), where  $c \in (0,1)$  is a positive constant and  $mean_A(\cdot)$  is the usual arithmetic mean.

Let  $f_{min} = \min_{x_i \in P} \{f(x_i)\}$ , and sort current population in their fitness ascending order  $f_{i,sort} = sort_{Ascending} \{f(x_i)\}_{x_i \in P}$ .

And the other parameters will be calculated by the following:

$$C_v(i) = \sin\left(\frac{f_{i,sort}}{N_p} \pi\right), \tag{15}$$

$$C_{R,iw}(k) = \frac{1 + \alpha_1 k}{1 + \alpha_2 k} \cdot \frac{\beta_2 + \beta_1 f_{min}}{1 + \beta_1 f_{min}} \cdot 0.041 \cdot C_v(i), \tag{16}$$

$$w_{i,d}(k) = \frac{1 + \alpha_1 k}{1 + \alpha_2 k} \cdot \frac{\beta_2 + \beta_1 f_{min}}{1 + \beta_1 f_{min}} \cdot (0.1rand(0,1) + 0.618), \tag{17}$$

$$p(k) = \frac{1 + \alpha_1 k}{1 + \alpha_2 k} \cdot \frac{\beta_2 + \beta_1 f_{min}}{1 + \beta_1 f_{min}} \cdot p_0, \tag{18}$$

where  $k$  is the iteration times or FEs,  $\alpha_1, \alpha_2, \beta_1, \beta_2$  are control parameters of the filter to be adjusted with iterations.

### 2.3 INERTIAL VELOCITY DIFFERENTIAL EVOLUTION ALGORITHM

Suppose that  $FEs$  is the variable of the objective function evaluations,  $FE_{smax}$  is the limit maximum of  $FEs$  for optimal problem to solve. The Inertial Velocity Differential Evolution (IVDE) algorithm is described as the following:

**Step 1:** Set the population size  $N_p$ , the dimension  $D$ , the value of  $FE_{smax}$ , the percentage  $p_0$  of  $\mathbf{x}_{best}^p$  in (18), the parameter value of  $\alpha_1, \alpha_2, \beta_1, \beta_2$  used in (16)~(18).

**Step 2:** Generate random initialization population  $P = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ , calculate their fitness and sort in ascending order, and set iteration control variable  $FEs = N_p$ .

**Step 3:** Set  $i=1$ .

**Step 4:** Select individual  $\mathbf{x}_i$  in current population, calculate the inertial velocity weighting factor  $w_{i,d}$  and crossover probability  $C_{R,iw}$  by (17) and (16) respectively. Then calculate  $C_R$  by (13) and  $F_i$  by (10).

**Step 5:** Calculate the variation  $\mathbf{v}_i$  of individual  $\mathbf{x}_i$  by (9) and (8), then determine an individual  $\mathbf{u}_i$  by (4) and (5).

**Step 6:** Calculate the fitness of the individual  $\mathbf{x}_i$ , and select the optimum assignment from  $\{\mathbf{u}_i, \mathbf{x}_i\}$  according to greedy selection mechanism by (6), update individuals to become the next generation of the population, and record the successful individuals to external storage.

**Step 7:**  $i=i+1$ . If  $i > N_p$ , go to Step 8; otherwise, go to Step 4.

**Step 8:** Set  $FEs = FEs + N_p$ .

**Step 9:** Record all individuals as the current population, calculate the  $\mu_F$  by (11),  $F_i$  by (10),  $\mu_{CR}$  by (14).

**Step 10:** Sort current population according to their fitness in ascending order, randomly select one of the top 100 p% individuals as  $\mathbf{x}_{best}^p$ .

**Step 11:** When  $FEs < FE_{smax}$ , go to Step 3; otherwise, the algorithm stops, the best individual will be regarded as the optimal solution.

To apply the IVDE algorithm to solve the TSP, the state vector and the individual fitness function should be defined corresponding to the TSP.

### 3 Task scheduling problem

The model of TSP in multiprocessor systems in this work can be described as follows [2]: multiprocessor system consists of  $M$  identical processors (their processing speeds are equal). The processors are fully connected and without communication delays. Each processor can execute at most one job at a time. Run-time of jobs may not be the same. A parallel program is represented by a direct acyclic graph  $G=(T,A,S,E)$ .  $T=(T_i), i=1,2,\dots,N$ , represents a set of tasks with associated weights ( $r_i$ ), where  $r_i$  denotes the duration of the execution of the job  $T_i$ .  $A=(a_{ij})$  is the set of

directed edges which define a partial order or limit order when preceding the execution of tasks set  $T$ .  $A$  is the set of arcs representing the activities such as communication delay,  $(i,j) \in A$  is the arc of the acyclic graph  $G$  from node  $i$  to  $j$  and there is only one directed arc  $(i,j)$  from  $i$  to  $j$ ,  $S \in T$  is the start node, and  $E \in T$  is the end node. Let each activity duration time be a stochastic variable denoted  $\xi = \{\xi_{ij}, (i,j) \in A\}$ , where  $\xi_{ij}$  are random duration times of activity represented by  $(i,j)$  in  $A$ . When  $T_i$  and  $T_j$  is scheduled on the same processor or the activity cost ignored, may let  $\xi_{ij} = 0$ . And, the schedule by the decision vector is denoted  $x = (x_1, \dots, x_i, \dots, x_D)$ , where  $x_i$  is a decision variable. It is assumed that all the decision variables may be translated into nonnegative integers  $j$  which denote  $i$ -th task is scheduled in  $j$ -th processor. Any task cannot start until all parents have finished. We denote  $TS_i(\xi, x)$  and  $TF_i(\xi, x)$  as the starting time and the finish time of all activities represented by  $(i,j)$  in  $A$  respectively. The starting time of the total task can be known as  $TS_S = TF_0(\xi, x) = 0$ , then

$$TS_j(\xi, x) = \max_{(i,j) \in A} \{TS_i(\xi, x) + r_i + \xi_{ij}\}. \quad (19)$$

After all tasks have been scheduled, the schedule length is defined as

$$TF_E(\xi, x) = TS_E(\xi, x) = \max_{(i,E) \in A} \{TS_i(\xi, x) + r_i + \xi_{iE}\}. \quad (20)$$

Task scheduling goal is to minimize the entire execution time of the task schedule  $x$ . That is

$$J(\xi, x) = \min \{TF_E(\xi, x)\}. \quad (21)$$

In the simplified case of communication problems between the two jobs is negligible, ie  $\xi_{ij} = 0$ . Jobs without predecessors are called entry tasks and tasks without successors are output operations. Each work is presented and appears exactly once in the schedule (completeness and uniqueness). An example of graph operations is shown in Figure 1 in the rectangle. There are 10 jobs. The circle is added to the index of jobs, the value on the right side of the circle is the task's duration. Preceding relation is shown directed lines (arrows). For example, the execution of work with index 4 is necessary to finish the job with the second index. Work with indices 8 preceding operations with index 1 and 5. The problem of optimal scheduling graph operations on a multiprocessor system with 3 identical processors consists of assigning business processes so that all tasks are completed in the shortest time [2].

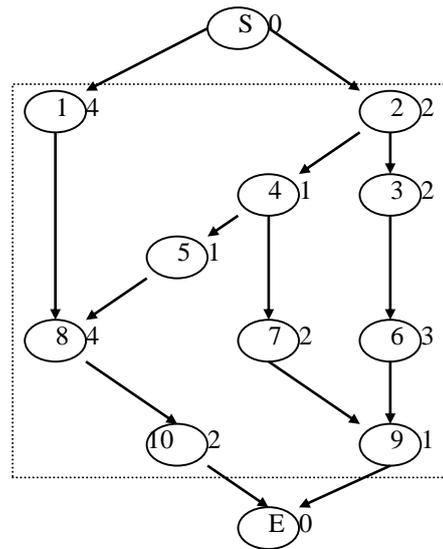


FIGURE 1 Extended graph operations from example [2]

#### 4 IVDE to solve TSP

Our scheduling goal is to minimize the entire execution time of the task schedule in  $M$  processors. Since the full search (Eng. exhaustive search) space solutions often impractical in most work explores the fast heuristic methods [1,2]. Heuristics does not guarantee finding the optimum, but can quickly find a solution that is close to optimal. Most heuristic methods of scheduling activities is one of the group scheduling lists. After all the work assigned to the priority they are added to the list of non-allocated jobs, which is sorted by priority drop-down. Since the end with the execution of assigned tasks, is selected from the list of job with highest priority and assigns the most appropriate processor. Algorithms in this group differ in accordance with the strategy of assigning priorities and the way it selects the most appropriate processor.

##### 4.1 REORDER THE TASKS

Reorder the tasks and rename their subscript to from the Task List according to their precedence relationship, make sure that if  $i < j$ , then  $TS_i \leq TS_j$ . This can be done from the start point  $S$  in a direct acyclic graph  $G=(T,A,S,E)$  by Breadth-First-Search algorithm.

##### 4.2 PRECEDENCE TASK ORDER CODER FOR IVDE

When using IVDE to solve TSP, we will define individual state as a task vector which dimension is equal to the number of tasks. For every individual, the precedence order of its vector value is a task scheduling sequence shown in Figure 2. For example, a ten dimension vector  $x = \{-9.76, 0.388, 2.69, 9.74, 7.45, -7.58, 1.66, 9.08, -6.84, 1.25\}$  can be used to solve the TSP shown in Figure 1. Because in the solution space,  $x_{i,\min} = -10$ ,  $x_{i,\max} = 10$ , we

can translate  $x_i$  into nonnegative integers  $j$  which denote  $i$ -th task is scheduled in  $j$ -th processor by the following

$$j = \text{round}((x_i - x_{i,\min}) \cdot M / (x_{i,\max} - x_{i,\min}) + 0.5) \quad (22)$$

where  $\text{round}(x)$  is a function to round the elements of  $x$  to the nearest integers. The vector  $x$  mentioned above is translated into  $\{1,2,2,3,3,1,2,3,1,2\}$ . And this can be regarded as the processor Priority List in the scheduling. So the task scheduling is shown in Figure 3. In the process of searching for the optimal solution, the individual can find the optimal precedence ordering by adjusting its value.

$x$	$x_1$	$x_2$	$x_3$	...	$x_i$	...	$x_N$
$T$	$T_1$	$T_2$	$T_3$	...	$T_i$	...	$T_N$

FIGURE 2 The relationship between task scheduling and IVDE individual

t	1	2	3	4	5	6	7	8	9	10	11
P <sub>1</sub>	T1				T6			T9			
P <sub>2</sub>	T2	T3			T7				T10		
P <sub>3</sub>			T4	T5	T8						

FIGURE 3 Task scheduling for problem on Figure 1 by IVDE

### 4.3 FITNESS FUNCTION OF IVDE TO SOLVE TSP

As shown in Figure 3, a vector  $x$  is a task schedule sequence, which give a schedule diagram, so we can use the maximum of the entire execution time of the schedule diagram to establish fitness function. Suppose give a  $x$ , the task precedence matrix is  $A=(a_{ij})$ , consider the task graphs without communication costs. Namely, let  $\xi_{ij} = 0$ . The fitness function of the individual  $x$  based on (21) is calculated as Figure 4.

```

Set N denotes tasks, M denotes processors
Translate x into integer vector J by Equation (22)
Define PFT(j) as j-th processor finish time
Define FT(i) as i-th task finish time
Define T(i) as i-th task executing time
Define A(i,j)=1 as i-th task is precedent of j-th task
For i=1 to N
    j=J(i)
    FT(i)=PFT(j)+T(i)
    If i>1
        For q=1 to i-1
            If A(q,i)=1
                If FT(q)>PFT(j)
                    PFT(j)=FT(q)
                    FT(i)=PFT(j)+T(i)
                End
            End
        End
    End
    PFT(j)=FT(i)
End
Select the maximum of PFT as the fitness of x
    
```

FIGURE 4 Fitness calculation algorithm in IVDE

### 4.4 THE COMPLEXITY OF IVDE TO SOLVE TSP

The complexity of the IVDE to solve TSP is mainly determined by the fitness function calculation. Shown in Figure 4, the fitness of  $x$  is set as the maximum  $PFT$  from  $M$  processor finish time of all tasks. The complexity of calculating  $PFT$  is  $O(N^2)$ . After that, the complexity of finding the maximum among  $PFT$  is  $O(M)$ . Therefore the fitness function calculation complexity is  $O(N^2)$ . IVDE can be used to solve TSP to meet the task scheduling analysis requirement of a complex system.

### 5 Numerical experiments

IVDE algorithm is programmed in Matlab environment. And the parameters of IVDE algorithm are set as:  $D=N$ =tasks,  $FE_{smax}=10^5$ ,  $N_p=100$ ,  $\lambda=1.1233$ ,  $c=0.1$ ,  $p_0=0.35$ ,  $\alpha_1=0.0000165$ ,  $\alpha_2=0.0001$ ,  $\beta_1=50$ ,  $\beta_2=0.5$ . The initial value of  $\mu_{CR}$  and  $\mu_F$  are 0.5.

As shown in Figure 3, the task scheduling problem is solved by IVDE. The minimum time of all task finished in 3 processors is 10. This is equal to the global optimum of this problem.

To further evaluate our proposed algorithms, we apply IVDE to solve some task graphs of specific benchmark application programs which are taken from a Standard Task Graph (STG) archive [4]. We first select three program used in [10] to test IVDE. The first programs of this STG set consists of task graphs generated randomly(tasks=100,rand000.stg) named Pg1, the second program is the robot control named Pg2 (tasks=90, robot.stg) as an actual application program and the last program is the sparse matrix solver named Pg3 (tasks=98, sparse.stg). However, we have not considered the task graphs with random communication costs. Namely, let  $\xi_{ij} = 0$ . The population size is considered to be 100, and the number of generations is considered to be 1000 generations. The processors is set  $M=4$ . The optimal solution is shown in Table 1 compared with the results in [10].

TABLE 1 Comparison with GA Algorithm in [10]

Application	Pg1 (Random graph)	Pg2 (Robot Control)	Pg3 (Sparse Matrix Solver)
SGA1[10]	301.6	1331.6	585.8
SGA2[10]	283.7	969	521.8
CPGA1[10]	183.4	848.5	301.8
CPGA2[10]	152.3	826.4	293.8
IVDE	149	781	486

In Table 1, SGA1 and SGA2 stand for Standard Genetic Algorithm with Roulette wheel selection and Tournament selection respectively; CPGA1 and CPGA2 stand for Critical Path Genetic Algorithm with Random order and ALAP order respectively in [10]. IVDE has gotten the best result for Pg1 and Pg2. For Pg3, the total tasks finished in one processor need cost of 1936. If there

are four processors to finish these tasks, the ideal minimum finished time is one fourth of 1936 in theory, namely 484. Therefore the optimal results of CPGA1 and CPGA2 are unreasonable. And IVDE has gotten the result of 486 which is very close to 484. This means that it is a good result.

TABLE 2 Comparison with other algorithm [12]

STG	Tasks	Processors	PDF/IHS	Ant Colony[12]	IVDE
Rand0008	50	2	281	281	281
Rand0038	50	4	114	114	114
Rand0107	50	8	155	155	155
Rand0174	50	16	131	131	131
Rand0017	100	2	569	569	569
Rand0066	100	4	253	253	<b>256</b>
Rand0106	100	8	205	205	205
Rand0174	100	16	162	162	152
Rand0020	300	2	827	<b>846</b>	<b>835</b>
Rand0095	300	8	382	<b>394</b>	<b>385</b>
Rand0136	300	16	324	<b>339</b>	324

For further research purposes a set of graphs is utilized from the website below: <http://www.kasahara.elec.waseda.ac.jp/schedule/index.html> [4]. Task graphs made available therein were divided into groups because of the number of tasks. Minimum scheduling length is calculated by means of PDF/HIS algorithm (Parallelized Depth First/Implicit Heuristic Search) for every tasks graph. Different task performance times, discretionary sequence constraints as well as random number of processors cause STG tasks scheduling problems to be NP-complete problems. Out of all solved problems heuristic algorithms under research did not find an optimal solution (assuming

this is the solution obtained with PDF/IHS algorithm) only for three of them. So, a parameter, called the relative error, is denoted as the error index and defined as  $|actual\ value - optimal\ value| / optimal\ value \times 100\%$ . However, results obtained are satisfactory, because the deviation from optimum varies from 0.96% to 1.19% which are better than the result obtained with Ant Colony optimizer in Table 2.

## 6 Conclusion

We have developed a new differential evolution algorithm with additional inertial velocity factor called Inertial Velocity Differential Evolution (IVDE) and proposed a new fitness function to solve the task scheduling problems for large scale system. We demonstrated our algorithms on STG set and get a better solution compared with the other heuristic algorithm in the paper [10, 12]. Conducted research shows that presented algorithms for task scheduling obtain good solutions irrespectively of investigated problem complexity. These solutions are considered optimal or sub-optimal whose deviation from optimum does not exceed 2%. Heuristic algorithms proposed for task scheduling problems, especially IVDE, should be a good tool for supporting planning process.

## Acknowledgments

The research is supported by the National Natural Science Foundation of Jiangxi Province, China (Grant No. 20132BAB201044) and Jiangxi Higher Technology Landing Project (Grant No.KJLD12071).

## References

- [1] Lee Y, Chen C 2003 A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems *Compiler Techniques for High-Performance Computing* URL: <http://parallel.iis.sinica.edu.tw/cthpc2003/> 16 Jan 2014
- [2] Golub M, Kasapović S 2002 Scheduling Multiprocessor Tasks With Genetic Algorithms *Proceedings of the IASTED International Conference Innsbruck* 273-8
- [3] Correa RC, Ferreira A, Rebreyend P 1999 Scheduling Multiprocessor Tasks with Genetic Algorithms *IEEE Transactions on Parallel and Distributed systems* 8 825-37
- [4] Advanced Computing Systems. Standard Task Graph Set URL: <http://www.kasahara.elec.waseda.ac.jp/schedule/> 1 Jul 2014
- [5] Markenscoff P, Joe D 1990 Computation of Tasks Modeled by Directed Acyclic Graphs on Distributed Computer Systems: Allocation Without Subtask Replication *IEEE Int'l Symp on Circuits & Sys* 2400-4
- [6] Lee C, 1992 Optimal Task Assignment in Linear Array Networks *IEEE Trans. on Computers* 41(7)
- [7] Nanda A, et. al 1992 Scheduling Directed Task Graphs on Multiprocessors Using Simulated Annealing *Proc. Int'l Conf. on Dist. Sys* 20-7
- [8] Lu M, Lam H C, Dai F 2008 Resource-constrained Critical Path Analysis based on discrete event Simulation and Particle Swarm Optimization *Automation in Construction* 17 670-81
- [9] Chang C K, Jiang H, Di Y, Zhu D, Ge Y 2008 Time-line Based Model for Software Project Scheduling with Genetic Algorithms *Information and Software Technology* 11 1142-54
- [10] Omara Fatma A, Arafa Mona M 2010 Genetic algorithms for task scheduling problem *Journal of Parallel and Distributed Computing* 70(1) 13-22
- [11] Montgomery J, Fayad C, Petrovic S 2006 Solution representation for job shop scheduling problems in ant colony optimization *LNCS* 4150 484-91
- [12] Helio J C, Barbosa 2013 Ant Colony Optimization - Techniques and Applications *ISBN 978-953-51-1001-9, Published: InTech* 212
- [13] Zhang J, Sanderson A C 2009 JADE: adaptive differential evolution with optional external archive *IEEE Trans Evolut Comput* 13(5) 945-58
- [14] Wang Y, Cai Z, Zhang Q 2011 Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters *IEEE Trans Evolutionary Computation* 15(1) 55-66
- [15] Kennedy J, Eberhart R C 1995 Particle swarm optimization [C] *Proceedings of IEEE Conference on Neural Networks IV, Piscataway, NJ IEEE Press* 1942-8

Authors	
	<p><b>Xiaohong Qiu, Jiangxi, China.</b></p> <p><b>Current position, grades:</b> professor of Software School, Jiangxi University of Science and Technology.</p> <p><b>University studies:</b> Engineering Doctor of Vehicle Control Guidance and Simulation. In 1995 graduated from the Department of Automation, Beijing University of Aeronautics and Astronautics.</p> <p><b>Scientific interest:</b> intelligent control and intelligent computing.</p> <p><b>Publications:</b> about 65 papers.</p>
	<p><b>Yuting Hu, October 05, 1990, Jiangxi, China.</b></p> <p><b>Current position, grades:</b> Post-graduate student in the Jiangxi University of Science and Technology.</p> <p><b>University studies:</b> Bachelor of Engineering in School of Electrical Engineering and Automation ,Jiangxi University of Science and Technology (2008-2012)</p> <p><b>Scientific interest:</b> intelligent computing.</p>
	<p><b>Bo Li, July 08, 1979, Jiangxi, China.</b></p> <p><b>Current position, grades:</b> lecturer of Software School, Jiangxi University of Science and Technology.</p> <p><b>University studies:</b> Master of Engineering in School of Science, Jiangxi University of Science and Technology (2002-2005).</p> <p><b>Scientific interest:</b> intelligent computing.</p>