Task Scheduling in Grid Computing Environments

Yi-Syuan Jiang and Wei-Mei Chen*

Department of Electronic and Computer Engineering,
National Taiwan University of Science and Technology,
Taipei 106, Taiwan
{M9902144,wmchen}@mail.ntust.edu.tw

Abstract. A grid computing environment is a parallel and distributed system that brings together various computing capacities to solve large computation problems. Task scheduling is a critical issue for grid computing, which maps tasks onto a parallel and distributed system for achieving good performance in terms of minimizing the overall execution time. This paper presents a genetic algorithm to solve this problem for improving the existing genetic algorithm with two main ideas: a new initialization strategy is introduced to generate the first population of chromosomes and the good characteristics of found solutions are preserved for new generations. Our proposed algorithm is implemented and evaluated using a set of well-known applications in our specific-defined system environment. The experimental results show that the proposed algorithm outperforms other algorithms within several parameter settings.

1 Introduction

In past few years, grid computing systems and applications become popular [3], due to a rapid development of many-core. A grid computing environment is a parallel and distributed system that brings together various computing capacities to solve large computation problems. In grid environments, task scheduling, which plays an important role, divides a larger job into smaller tasks and maps tasks onto a parallel and distributed system [1,6]. The goal of a task scheduling is typically to schedule all the tasks on a given number of available processors so as to minimize the overall length of time required to execute the whole program.

A parallel and distributed computing system may be homogeneous [10] or heterogeneous systems [7,11,13]. A homogeneous system means that the processors are the same performance in processing capabilities. On the other hand, heterogeneous systems have different processing capabilities in the target system. In general, the processors are connected by an interconnection network, which is either fully-connected [11,13] or partially-connected [2]. In the fully-connected network every processor can communicate with each other, whereas data can be transferred to some specified processors in a partially-connected network. Besides, the task duplication issue [10] was also discussed to reduce the communication time by duplicating some tasks on more than one processor to eliminate communication cost. To avoid increasing energy consumption,

* This work is partially supported by National Science Council under the Grant NSC 101-2221-E-011-039-MY2.

here we consider the target system which is the fully-connected heterogeneous systems without task duplication.

The genetic algorithm (GA), first proposed by Holland [5], provides a popular solution for application problems [4,9]. GAs have been shown that outperforms several algorithms in the task scheduling problem, which simply define the search space to be the solution space in which each point is denoted by a number string, called a chromosome. Based on these solutions, three operators which are selection, crossover, and mutation, are employed to transform a population of chromosomes to better solutions iteratively. In order to keep the good features from the previous generation, the crossover operator exchanges the information from two chromosomes chosen randomly, and the mutation operator alters one bit of a chromosome.

In this paper, we proposed a genetic algorithm for task scheduling on a grid computing system, called TSGA. In general, GA approaches directly initialize the first population by some uniform random process. TSGA develops a new initialization policy, which divides the search space into specific patterns in order to accelerate the convergence of solutions. To solve the task scheduling problem, a chromosome usually contains a mapping part and an order part to indicate the corresponding computer and the executing order. In the standard GA, when crossover and mutation operators are applied, both of the mapping part and the order part will be changed, which brings that the parents' characteristics cannot be kept in the next generation. Inspired by the idea of eugenics, TSGA presents new operators for crossover and mutation to preserve good features from the previous generation.

The remainder of the paper is organized as follow. In the next section, we provide the problem definition. The proposed genetic scheduling algorithm is presented in Section 3. We describe our experimental results in Section 4. Finally, conclusions are drawn in Section 5.

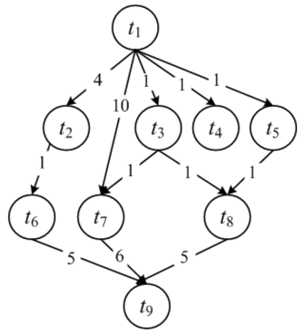
2 Problem Definition

Task scheduling is mapping smaller tasks to multiprocessors. Tasks with data precedence are modeled by a Directed Acyclic Graph (DAG) [13]. The main idea of DAG scheduling is minimizing the makespan which is the overall execution time for all tasks.

2.1 DAG Modeling

A DAG $G = (V,E)$ is depicted in Fig. 1(a), where V is a set of N nodes and E is a set of M directed edges. For the problem of task scheduling, V represents the set of tasks and each task contains a sequence of instructions that should be completed in a particular order. Let $w_{i,j}$ be the computation time to finish a particular task $t_i \in V$ on the processor P_j, detailed in Fig. 1(b). Each edge $e_{i,j} \in E$ in the DAG indicates the precedence constraint that task t_i should complete its execution before task t_j starts. Let $c_{i,j}$ denote the communication cost needed to transport the data between task t_i and task t_j, which is the weight on an edge $e_{i,j}$. If t_i and t_j is assigned to the same processor, the communication cost $c_{i,j}$ is zero.

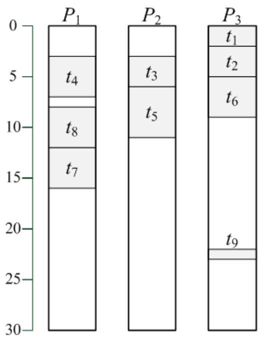
The source node of an edge is called a predecessor of that node. Similarly, the destination node emerged from a node is called a successor of that node. In Fig. 1(a), t_1 is the predecessor of t_2, t_3, t_4, and t_5. On the other hand, t_2, t_3, t_4, and t_5 are the successor of t_1. In a graph, a node with no parent is called an entry node, and a node with no child is called an exit node. If a node t_i is scheduled to a processor P_j, the start-time and the finish-time of t_i are denoted by $ST(t_i, P_j)$ and $FT(t_i, P_j)$, respectively.



(a)

task	P_1	P_2	P_3
1	3	1	2
2	3	3	3
3	4	3	2
4	4	2	6
5	7	5	3
6	2	6	4
7	4	6	2
8	4	3	5
9	1	1	1

(b)



(c)

Fig. 1. An example of (a) DAG, (b) the computation cost matrix, and (c) an example of scheduling

2.2 Makespan

After all tasks are scheduled onto parallel processors, considering a particular task t_i on the processor P_j, the start-time $ST(t_i, P_j)$ can be defined as

$$ST(t_i, P_j) = \max\{RT_j, DAT(t_i, P_j)\},$$

where $DAT(t_i, P_j)$ is the data arrival time of task t_i at the processor P_j, which is the time when all the needed data have been transmitted. On the other hand, $DAT(t_i, P_j)$ is defined as

$$DAT(t_i, P_j) = \max_{t_k \in pred(t_i)} \{(FT(t_k, P_j) + c_{k,i})\},$$

where $pred(t_i)$ denotes the set of immediate predecessor tasks of the task t_i. Since

$$FT(t_k, P_j) = w_{k,j} + ST(t_k, P_j)$$

and

$$RT_j = \max_{t_k \in exe(P_j)} \{FT(t_k, P_j)\},$$

where $exe(P_j)$ is the set containing tasks which executes on the processor P_j, the overall schedule length of the entire program is the largest finish time among all tasks and can be expressed as

$$makespan = \max_{t_i \in V} \{FT(t_i, P_j)\}.$$

Fig. 1(c) demonstrates a scheduling for the graph described in Fig. 1(a). The makespan of this scheduling is 23.

3 Proposed Method

In this section, we introduce TSGA algorithm in detail, including the encoded and decoded representations and five important operators.

3.1 The Representation of Solutions

The representation of a chromosome is given in Fig. 2, which is divided into a mapping part (S_M) and an order part (S_O). We use integer arrays to store S_M and S_O and the size of arrays is equal to the number of tasks. If $S_M[i]$ is j and $S_O[i]$ is k, it means that a task t_k is executed on the processor P_j.

According to the chromosome represented in Fig. 2, the solution of a DAG in Fig. 1(a) can be scheduled in Fig. 1(c). First, we assign tasks into the mapping processor according to the index of S_M. Tasks t_4, t_7, and t_8 are scheduled on processor P_1. Tasks t_3, and t_5 are executed on processor P_2. Tasks t_1, t_2, t_6, and t_9 are assigned to the processor P_3. Following the order in S_O, we schedule t_4, t_7, and t_8 in the order of t_4, t_8, t_7 in P_1. For P_2, t_3 is executed before t_5. Tasks t_1, t_2, t_6, and t_9 are taken in the order of t_1, t_2, t_6, t_9 in P_3. Finally, we should count the wait time for communicating, if two dependent tasks are scheduled on a different processor.

TSGA defines the fitness function in order to measure the quality of solutions. The purpose of the scheduling problem is minimizing the makespan. Thus, the fitness function is defined as the makespan.

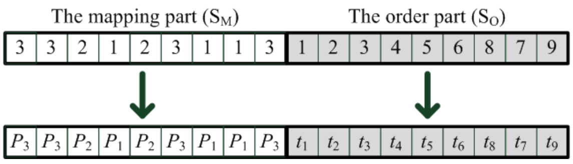


Fig. 2. A representation of chromosome

3.2 TSGA

An algorithmic flowchart of TSGA is given in Fig. 3. If the number of generation is not smaller than the maximum generation G, TSGA will output the best solution.

Initialization. As shown in Algorithm 1, TSGA initializes the first population that consists of encoded chromosomes. Each chromosome is composed of the processor assignment and the execution order. Instead of using the random strategy to give the processor assignment, we devise a new method dividing the search space into specific patterns equally. The search space is divided into $\log_2 n$ subspaces, where n is the number of processors. Since the best scheduling solution may occupy either few processors or most processors in different cases, we give some patterns with a different number of processors in order to explore the solution space in different aspects.

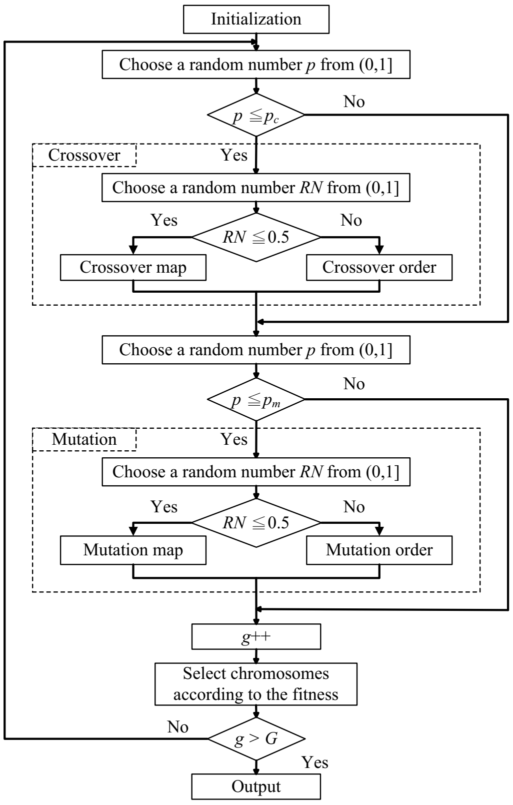

Initialization
Choose a random number p from (0,1]
$p \leqq p_c$
No
Crossover
Yes
Choose a random number RN from (0,1]
Yes
No
$RN \leqq 0.5$
Crossover map
Crossover order
Choose a random number p from (0,1]
$p \leqq p_m$
No
Mutation
Yes
Choose a random number RN from (0,1]
Yes
No
$RN \leqq 0.5$
Mutation map
Mutation order
g++
Select chromosomes according to the fitness
No
$g > G$
Yes
Output

Fig. 3. TSGA flowchart

Crossover Map Operator. As shown in Fig. 4(a), the crossover map operator is used for changing the mapping processor of two chromosomes. The crossover map operator chooses two chromosomes S and T from the population and an integer I between 1 and N randomly. TSGA keeps the processor assignment which is located on the left of I. For the processors on the right of I, we exchange the processors of S and T which are assigned to execute the same task. The processor assignments of tasks t_5, t_6, and t_9 are exchanging directly, since those tasks are occupied in the same place in both chromosomes. On the other hand, tasks t_7 and t_8 are located at different places in these two chromosomes, so they are scheduled to the processor in which they are assigned to another chromosome, and TSGA exchanges the processor assignments. The chromosome S'' and T'' are generated by the crossover map operator in SGA.

Crossover Order Operator. The crossover order operator is practiced to the second part of the chromosome. In Fig. 4(b), after choosing two chromosomes S and T, TSGA chooses a crossover point I between 1 and N. Then, Step 1 copies the left portion of I in S and T to the new chromosome S' and T', respectively. In order to complete the right

Algorithm 1. Initialization_operator(C, n, DAG)

```
Require: C: population set; n: the number of processors
Ensure: C
 1: $d \leftarrow \log_2 n$   {Divide chromosomes into $d$ groups}
 2: for group =1 to $d$ do
 3:     for $i = 1$ to population_ size$/d$ do
 4:         $S_M \leftarrow$ choose a processor for each task from $\{1, 2, \ldots, 2^d\}$
 5:         $S_O \leftarrow$ an executed order according to a topological ordering
 6:         $S \leftarrow S_M \oplus S_O$
 7:         $C \leftarrow C \bigcup \{S\}$
 8:     end for
 9: end for
10: return C
```

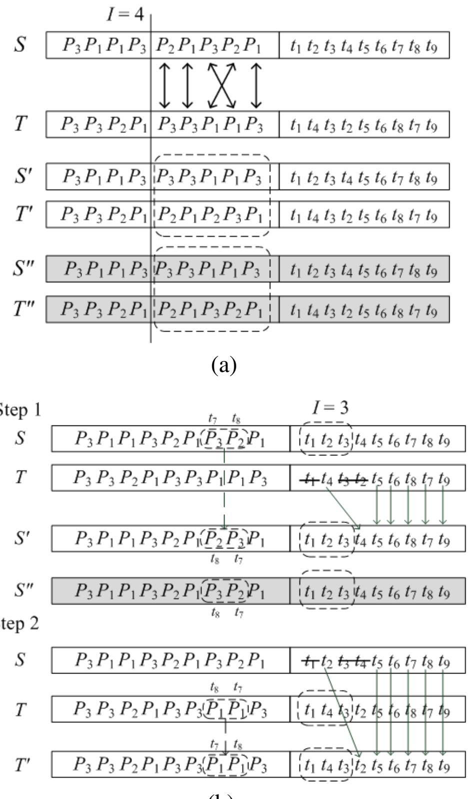
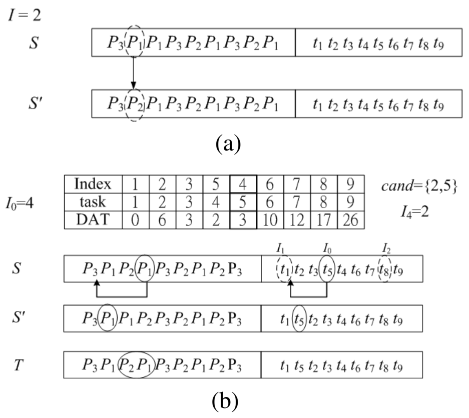


Fig. 4. An example of (a) the crossover map operator and (b) the crossover order operator

segment of I, the crossover order operator carries tasks into S' according to the order in T. Step 2 uses the same rules in step 1 to generate the second offspring T'. Note that the processor assignment should be adjusting to corresponding to the original processor assignment. The chromosome S'' is created by the crossover order operator in SGA.


I = 2
S
P3 P1 P1 P3 P2 P1 P3 P2 P1 | t1 t2 t3 t4 t5 t6 t7 t8 t9
S'
P3 P2 P1 P3 P2 P1 P3 P2 P1 | t1 t2 t3 t4 t5 t6 t7 t8 t9
(a)
I0=4
cand={2,5}
I4=2
I1 I0 I2
S
P3 P1 P2 P1 P3 P2 P1 P2 P3 | t1 t2 t3 t5 t4 t6 t7 t8 t9
S'
P3 P1 P1 P2 P3 P2 P1 P2 P3 | t1 t5 t2 t3 t4 t6 t7 t8 t9
T
P3 P1 P2 P1 P3 P2 P1 P2 P3 | t1 t5 t2 t3 t4 t6 t7 t8 t9
(b)

Index	1	2	3	5	4	6	7	8	9
task	1	2	3	4	5	6	7	8	9
DAT	0	6	3	2	3	10	12	17	26

Fig. 5. An example of (a) the mutation map operator and (b) the mutation order operator

Table 1. Parameter settings used in the experiment

Population size	Generation size	Crossover rate	Mutation rate	Selection operator
400	1000	0.8	0.2	Binariesry tournament

Mutation Map Operator. The mutation map operator is used for the mapping section of the chromosome, shown in Fig. 5(a). Like the crossover operator, choosing a random number I between 1 to N is essential. Then, the mutation map operator changes the processor in which the chosen task $S_O[I]$ is executed to another processor chosen randomly.

Mutation Order Operator. The mutation order operator is applied to the second part of the chromosome. We select an index I_0 from 1 to N randomly, which decides a task $S_O[I_0]$ to will be mutated. We create an empty set *cand* which would contain indexes of tasks that can be inserted. I_1 and I_2 are the index of the first related task of task $S_O[I_0]$ on the left and right of I_0, respectively. For the index i between I_1+1 and I_0, if DAT of $S_O[i]$ is larger than DAT of $S_O[I_0]$, task $S_O[I_0]$ should be executed before task $S_O[i]$, since task $S_O[I_0]$ can be executed early. For the index i between I_0 and I_2-1, if DAT of $S_O[i]$ is smaller than DAT of $S_O[I_0]$, task $S_O[i]$ should be executed before task $S_O[I_0]$, since task $S_O[i]$ can be executed early. We add tasks which meets those conditions to the set *cand*. If *cand* is empty, we fill *cand* with indexes between I_1+1 and I_2-1. We select an index I_4 from *cand* randomly, and move task $S_O[I_0]$ to the location I_4. Finally, we adjust the processor assignment to correspond with the original processor assignment, for the same reason mentioned in the crossover order operator. Fig. 5(b) demonstrates the mutation order operator.

4 Experimental Results

This paper considers five well-known applications, Gauss-Jordan elimination (GJ) [14], the fast Fourier transformation (FFT) [11], Robot control, Sparse matrix solver, and

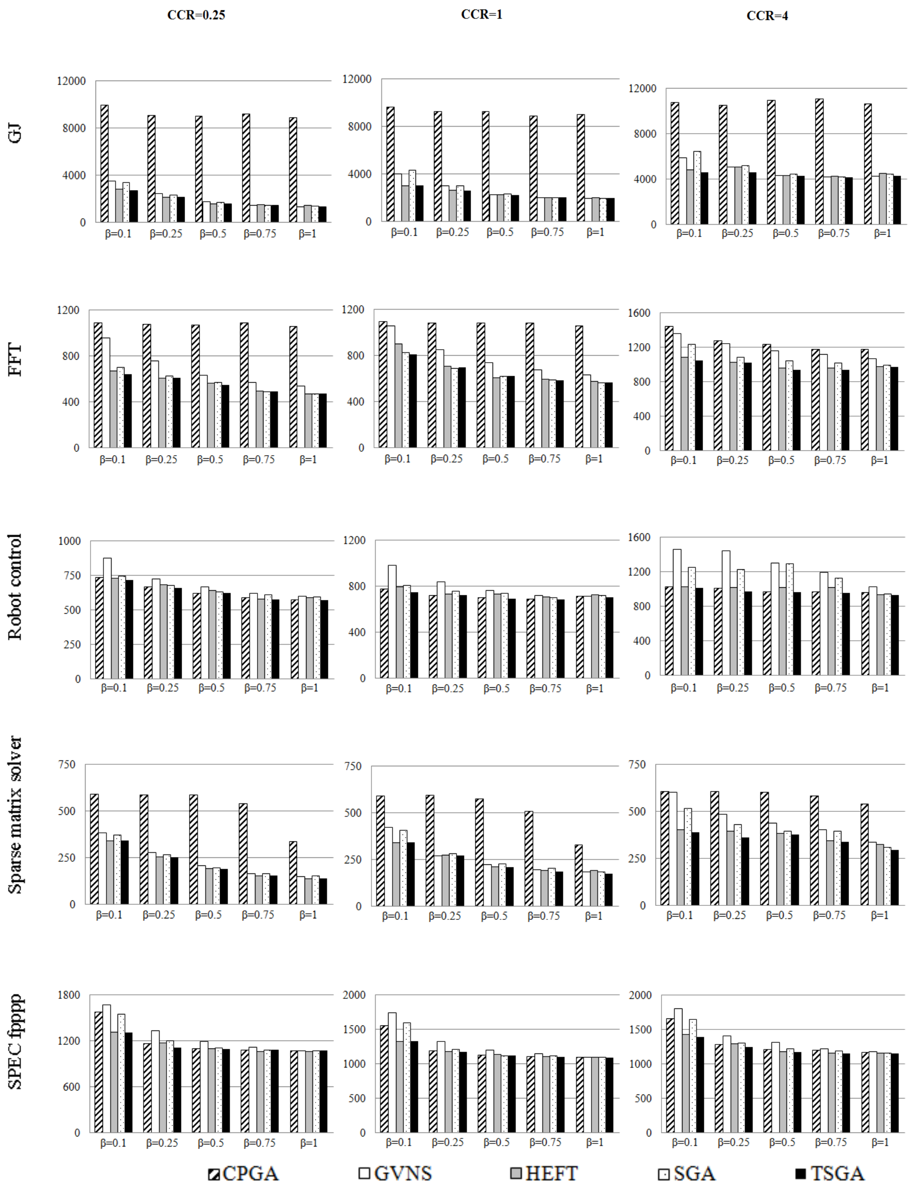


Fig. 6. Experimental results with varying parameters

SPEC fpppp are regared as the benchmark in the problem. Robot control, Sparse matrix solver, and SPEC fpppp are taken from a Standard Task Graph (STG) archive [15]. GJ, FFT, Robot control, Sparse matrix solver, and SPEC fpppp have 300, 223, 88, 96, and 334 tasks, and 552, 382, 131, 67, and 1145 edges, respectively. Each application has various features depending on different settings as below.

1) The number of processors (P):
   $SET_P = \{4, 8, 16\}$
2) The range percentage of computation costs on processors (β):
   $SET_\beta = \{0.1, 0.25, 0.5, 0.75, 1.0\}$
3) The communication to computation ratio (CCR):
   $SET_{CCR} = \{0.25, 0.5, 1.0, 2.0, 4.0\}$

To evaluate our proposed algorithms, we have implemented them using an AMD FX(tm)-8120 eight-core processor (3.10 GHz) using C++ language. Generally speaking, the excellent solution of various problems would be generated by various parameters for a specific algorithm. However, we use the same parameter values listed in Table 1, to show the performance in terms of makespan in this paper.

The performance of TSGA is compared with four algorithms, CPGA [10], SGA, GVNS [13], and HEFT [11]. For each data configuration of five DAGs, the average of makespan obtained over 10 runs is computed for CPGA, SGA, GVNS, and TSGA. On the other hand, HEFT is run only once, since it is a deterministic algorithm.

The experimental results of five DAGs with P fixed to 16 and varying β value are given in Fig. 6, which presents histograms in a table way. Column 1 indicates that a specific application was tested. The parameter CCR is recorded in the row 1, if the CCR value is changing. Every application, shown in each row, was tested with different CCR value and varying β value. Each column contains five applications with fixed CCR value. For instance, the result of testing FFT with $CCR = 4$ and varying β value is given in the row 3 and column 4, and so on. Note that, the vertical coordinate shows the makespan.

5 Conclusion

In this paper, we presented a genetic algorithm for task scheduling, called TSGA, to solve the problem of task scheduling on parallel and distributed computing systems by improving the standard genetic algorithm by increasing the convergence speed and preserving the good features of previous generation. To demonstrate the TSGA performance, we introduced new genetic operators in TSGA. The initialization operator divides the search space into specific patterns so that we can save much time to explore the whole search space. The crossover map operator and the mutation map operator help us to find more suitable processor assignments and the crossover order operator and the mutation order operator provide us with more efficient execution order. For some cases with high CCR values, TSGA schedules tasks to occupy fewer processors to reduce the communication time. For some cases with small β values, TSGA assigns tasks to occupy the processors which have higher processing capability to minimize the execution time. The experimental results show that the proposed algorithm outperforms other algorithms within several parameter settings.

References

1. Culler, D., Singh, J., Gupta, A.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publisher, San Francisco (1998)
2. Choudhury, P., Chakrabarti, P.P., Kumar, R.: Online Scheduling of Dynamic Task Graphs with Communication and Contention for Multiprocessors. IEEE Trans. on Parallel and Distributed Systems 23(1), 126–133 (2012)
3. Falzon, G., Li, M.: Enhancing Genetic Algorithms for Dependent Job Scheduling in Grid Computing Environments. Journal of Supercomputing 62(1), 290–314 (2012)
4. Han, Q., Yu, L., Zheng, W., Cheng, N., Niu, X.: A Novel QKD Network Routing Algorithm Based on Optical-Path-Switching. Journal of Information Hiding and Multimedia Signal Processing 5(1), 13–19 (2014)
5. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
6. Hwang, K.: Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill, Inc., New York (1993)
7. Kwok, Y.K., Ahmad, I.: Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm. Journal of Parallel and Distributed Computing 47(1), 58 (1997)
8. Kwok, Y.K., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
9. Loukhaoukha, K.: On the Security of Digital Watermarking Scheme Based on Singular Value Decomposition and Tiny Genetic Algorithm. Journal of Information Hiding and Multimedia Signal Processing 3(2), 135–141 (2012)
10. Omara, F.A., Arafa, M.M.: Genetic algorithms for task scheduling problem. Journal of Parallel and Distributed Computing 70(1), 13–22 (2010)
11. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing. IEEE Trans. on Parallel and Distributed Systems 13(3), 260–274 (2002)
12. Wu, A.S., Yu, H., Jin, S., Lin, K., Schiavone, G.: An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. IEEE Trans. on Parallel and Distributed Systems 15(9), 824–834 (2004)
13. Wen, Y., Xu, H., Yang, J.: A Heuristic-based Hybrid Genetic-variable Neighborhood Search Algorithm for Task Scheduling in Heterogeneous Multiprocessor System. Information Sciences 181(3), 567–581 (2011)
14. Yu, H.: Optimizing Task Schedules using an Artificial Immune System Approach. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 151–158 (2008)
15. `http://www.Kasahara.Elec.Waseda.ac.jp/schedule/`