Engineering Applications of Artificial Intelligence 61 (2017) 35–46

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems

Mehdi Akbari[a], Hassan Rashidi[b,*], Sasan H. Alizadeh[c]

[a] Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
[b] Department of Mathematics and Computer Science, Allameh Tabataba'i University, Tehran, Iran
[c] Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

ARTICLE INFO

Keywords:
Task scheduling
Heterogeneous systems
Genetic algorithm
Meta-heuristic algorithm

ABSTRACT

One of the important problems in heterogeneous computing systems is task scheduling. The task scheduling problem intends to assigns tasks to a number of processors in a manner that will optimize the overall performance of the system, i.e. minimizing execution time or maximizing parallelization in assigning the tasks to the processors. The task scheduling problem is an NP-complete and this is why the algorithms applied to this problem are heuristic or meta-heuristic by which we could reach a relatively optimal solution. This paper presents a genetic-based algorithm as a meta-heuristic method to address static task scheduling for processors in heterogeneous computing systems. The algorithm improves the performance of genetic algorithm through significant changes in its genetic functions and introduction of new operators that guarantee sample variety and consistent coverage of the whole space. Moreover, the random initial population has been replaced with some initial populations with relatively optimized solutions to lower repetitions in the genetic algorithm. The results of running this algorithm on the graphs of real-world applications and random graphs in heterogeneous computing systems with a wide range of characteristics, indicated significant improvements of efficiency of the proposed algorithm compared with other task scheduling algorithms.

1. Introduction

Heterogeneous distributed systems contain a group of varied-pace processors connected to each other through a fast network. They are employed for parallel execution of distributed applications. Task-scheduled algorithms help the processors to manage task completion via assigning tasks to available processors, thus enhancing the efficiency. They have been widely adopted in industrial and manufacturing applications (Savino et al., 2015, 2014a, 2014b, 2010). Task scheduling is an NP-complete (Ullman, 1975) and thus meta-heuristic algorithms are used to reach a relatively optimized solution, this means that to reach a solution close to the optimum solution. One of the main challenges faced by meta-heuristic algorithms such as genetic algorithms is the frequency of their repetitions for reaching relatively optimized solutions and also conflicting local optimums. Efficient genetic algorithm for task scheduling (EGA-TS) that is presenting in this paper tackles these issues via creating a new initial population function and adding improved operators to the basic genetic algorithm.

The main objective of the task scheduling algorithms is to achieve a balance between parallelization increase and communication costs decrease. EGA-TS primarily intends to minimize total execution time, assigning tasks to processors for a maximum of parallelization while minimizing communications costs in proportion to task execution costs of the processors. Lowering the repetition frequency and avoiding local optimums are also intended by this algorithm, this is achieved through minimizing repeated or similar schedules in the population. To guarantee sample variety and consistent coverage of the whole space, some significant changes are made in genetic functions and new operators are introduced. The major contributions of this study are listed below:

- In most of metaheuristic-based scheduling algorithms, there is a stage for selection and assignment of tasks to processors (Ahmad et al., 2016; Gogos et al., 2016; Kumar and Vidyarthi, 2016; Wang et al., 2016; Zhang et al., 2015). However, a new method has been developed in the proposed algorithm to display schedules in which each task is displayed in conjunction with the processor that was assigned in the stage of generating initial populations.
- In standard GA, initial population is generated randomly. Random treatment of initial population selection leaves us with unsuitable

* Corresponding author.
E-mail addresses: mehdi_akbari@hotmail.com, mehdi_akbari@pco.iaun.ac.ir (M. Akbari), Hrashi@atu.ac.ir (H. Rashidi), sasan.h.alizadeh@qiau.ac.ir (S.H. Alizadeh).

http://dx.doi.org/10.1016/j.engappai.2017.02.013
Received 24 March 2016; Received in revised form 24 February 2017; Accepted 26 February 2017
Available online 01 March 2017

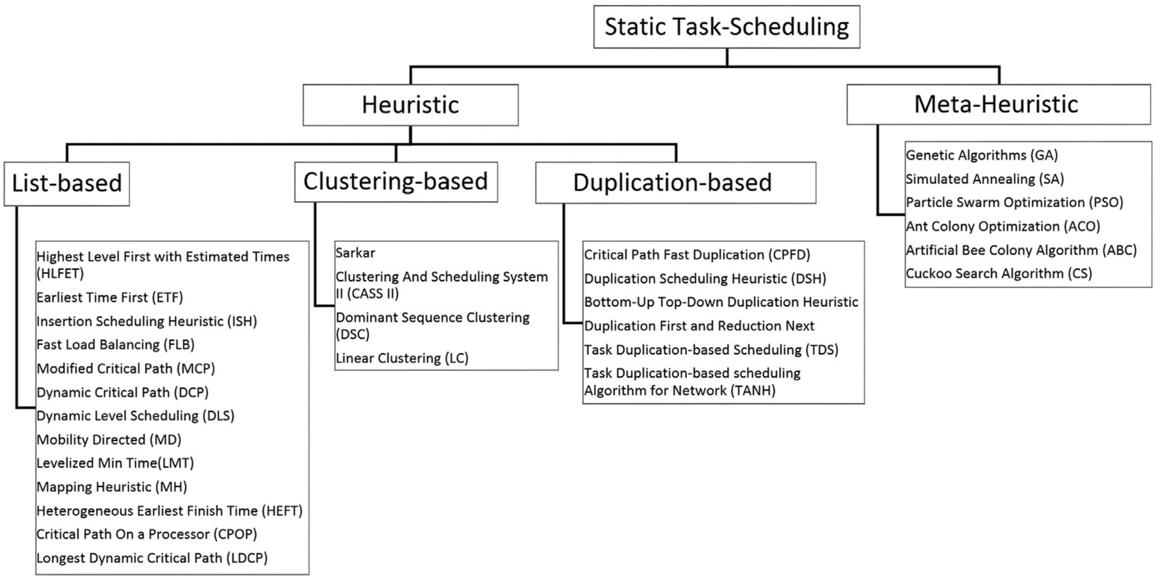

Static Task-Scheduling
Heuristic
Meta-Heuristic
List-based
Clustering-based
Duplication-based
Highest Level First with Estimated Times (HLFET)
Earliest Time First (ETF)
Insertion Scheduling Heuristic (ISH)
Fast Load Balancing (FLB)
Modified Critical Path (MCP)
Dynamic Critical Path (DCP)
Dynamic Level Scheduling (DLS)
Mobility Directed (MD)
Levelized Min Time(LMT)
Mapping Heuristic (MH)
Heterogeneous Earliest Finish Time (HEFT)
Critical Path On a Processor (CPOP)
Longest Dynamic Critical Path (LDCP)
Sarkar
Clustering And Scheduling System II (CASS II)
Dominant Sequence Clustering (DSC)
Linear Clustering (LC)
Critical Path Fast Duplication (CPFD)
Duplication Scheduling Heuristic (DSH)
Bottom-Up Top-Down Duplication Heuristic
Duplication First and Reduction Next
Task Duplication-based Scheduling (TDS)
Task Duplication-based scheduling Algorithm for Network (TANH)
Genetic Algorithms (GA)
Simulated Annealing (SA)
Particle Swarm Optimization (PSO)
Ant Colony Optimization (ACO)
Artificial Bee Colony Algorithm (ABC)
Cuckoo Search Algorithm (CS)

Fig. 1. Classification of static task scheduling algorithms at compile time.

search space. In suitable search space, the samples are close to the optimal solution. Therefore we should generate initial populations that allow us to reach relatively optimized solutions in reasonable runtime.

- The proposed algorithm presents the new operator which is called Inversion. This operator promotes diversity among the samples that will be produced in each generation. By using this new operator the number of repeated schedules in generated population is minimized.

The remaining parts of this paper is structured as follows. Section 2 presents the related works and then we proceed to explain application task graphs along task scheduling model in Section 3. Section 4 is dedicated to the improved genetic algorithm and drawing comparisons with some criteria. In Section 5 we evaluate the time and space complexities. Section 6 presents the results of the comparisons done and the simulations made. Conclusions are presented in the last section.

2. Related works

Static task scheduling algorithms are categorized as heuristic and meta-heuristic (Bansal et al., 2003; Kwok and Ahmad, 1996; Manudhane and Wadhe, 2013; Topcuoglu et al., 2002). Heuristic algorithms are in turn divided into list-based, clustering and duplication methods (Topcuoglu et al., 2002).

In list-based heuristics, a priority is assigned to each task and the tasks are listed in order of their preferences. In these kinds of heuristics, the task selection for processing is done in order of preference and a task with a higher priority is assigned to processor earlier. Heterogeneous earliest finish time (HEFT) and critical path on a processor (CPOP) are the most important examples of list based heuristics (Topcuoglu et al., 2002).

The HEFT algorithm is designed for a given number of heterogeneous processors and includes two stages: at the first stage, task scheduling priority is calculated and at the second stage, processor selection stage, tasks are analyzed in order of priority and assigned to a processor providing the shortest finish time. In this algorithm, priority is determined by a pair of parameters known as *tlevel* and *blevel*. In tasks graph, *tlevel* of each node denotes the longest path weight from the start node to the desired node. In another aspect, *tlevel* of each node represents the nearest possible start time of that node and *blevel* of each node shows the longest pass weight of that node to the exit node. If this algorithm uses *tlevel* parameter to determine the priorities it is called HEFT-T and if *blevel* parameter is applied it is called HEFT-B.

In CPOP algorithm, the total sum of *blevel* and *tlevel* of each node is regarded as task priority and then tasks are selected in order of priority and if placed on critical path, are allocated to the processor which minimizes total task calculation costs on critical path and if not, they are assigned to the processor which makes the nearest complete time possible for it. Critical path refers to the path from the input node to the output node that has the highest total value of calculation cost and communication costs of edges (Kwok and Ahmad, 1996). Therefore, an effective scheduling list-based algorithm requires an appropriate scheduling of tasks placed on critical path (Daoud and Kharma, 2011). The length of critical path is equal to *blevel* and *tlevel* sum of input task. Thus, each task whose *tlevel* and *blevel* equals to the sum of *blevel* and *tlevel* of input task lies on critical path (Topcuoglu et al., 2002).

Duplication heuristic method reduces runtime by applying task duplication to different processors (Lin et al., 2013). In this method, communication time between processors is reduced by executing tasks on more than one processor. It avoids the transmission of the results from a specific task to the next one (because both are executed on a single processor) and therefore it reduces communication costs. In parallel and distributed systems, clustering heuristic method is an appropriate solution to reduce graph communication delay. Communication delay is reduced in this method since tasks which have high relations with each other are put together in a cluster and are assigned to a single processor (Mishra et al., 2012). Fig. 1 shows this classification together with the proposed algorithms samples for each category (Topcuoglu et al., 2002).

Heuristic based algorithms are not distinctly possible to produce constant results for a wide range of problems, mostly when the complexity of the task scheduling problem increases. Contrary to the meta-heuristic algorithms, a heuristic algorithm uses a combinatory process in the search for solutions, which is less efficient and generates much higher computational cost than the meta-heuristic algorithms. For this reason, balance between makespan and speed of convergence is required.

The Meta-heuristic algorithms usually make up an important solution for global optimization problems. Totally heuristic means to find and discover by error and trial. "Meta-heuristic" can be applied to strategies with a higher level that have modified and guided heuristic methods in a way that can achieve solutions and innovations beyond what is normally accessible in local optimum search. There are two types, local and global optima while dealing with optimization problems. The local optima are the best solution found in a subset of solution spaces but not necessarily the best for the whole problem space. In contrast, the global optima are the best solution to the whole problem space. Finding the global optima in most real-life problems is extremely difficult and therefore satisfactory and good-enough solutions are often accepted. Meta-heuristic algorithms are employed to achieve these targets. The reasons for using meta-heuristic algorithms could be put in three categories of simplicity, flexibility and ergodicity which come as below:

- ***Simplicity***: Most meta-heuristic algorithms are simple, easy to implement and relatively less complex. The primary part of the algorithm could be written in 100 lines in programming languages.
- ***Flexibility***: These algorithms are flexible as they are simple to cover a wide range of optimization problems which can't be handled by classic algorithms.
- ***Ergodicity***: Meta-heuristic algorithms have high degrees of ergodicity which means they can search multi-modal search spaces with sufficient variety and avoiding local optima at the same time. Ergodicity is often the result of randomized techniques derived from natural systems such as crossover and mutation or statistical models such Random walks or Le'vy flights (Yang et al., 2013).

There are various methods to solve task scheduling problems based on meta-heuristic methods. The most famous ones are:

- ***Genetic based algorithms*** (Daoud and Kharma, 2011; Gupta et al., 2010; Hwang et al., 2006; Kołodziej and Khan, 2012; Lu et al., 2013; Omara and Arafa, 2010; Rahmani and Vahedi, 2008; Sathappan et al., 2011; Singh and Singh, 2012; Zomaya et al., 1999).
- ***Particle Swarm Optimization*** (PSO) (Guo et al., 2012; Zhang et al., 2008; Zuo et al., 2014).
- ***Ant Colony Optimization*** (ACO) (Babukartik and Dhavachelvan, 2012; Kim and Kang, 2011; Lo et al., 2008; Pendharkar, 2015; Yang et al., 2010).
- ***Cuckoo Search Algorithm*** (CS) (Akbari and Rashidi, 2016; Navimipour and Milani, 2015).
- ***Artificial Bee Colony*** (ABC) (Babukartik and Dhavachelvan, 2012; Ferrandi et al., 2010; Lin et al., 2014).
- ***Simulated Annealing algorithm*** (Damodaran and Vélez-Gallego, 2012; Wang et al., 2010),
- ***Memetic Algorithm*** (Pendharkar, 2011).

Table 1 provides a list of algorithms and their definitions as used in the paper. The time complexity of SA is O(τ×L×ln |R|) where τ is the time involved in the generation and (possible) acceptance of a transition, L is the size of the largest neighborhood and R is the set of configurations or solutions of the problem (Van Laarhoven et al., 1992). The time complexity of PSO is O(M×K×log K) where M is the number of objective functions and K is solutions in the archive (Bandyopadhyay et al., 2008; Raquel and Naval Jr, 2005)

Table 1
Definition of algorithms.

Algorithm	Type	Priority	Time complexity
HEFT-B	Static list scheduling +Insertion-based	Blevel	$O(n^2 \times p)$
HEFT-T	Static list scheduling +Insertion-based	Tlevel	$O(n^2 \times p)$
CPOP	Static list scheduling	blevel+tlevel	$O(n^2 \times p)$
BGA	Genetic	blevel+tlevel	$O(\text{generations} \times n^2 \times p)$
SA	Simulated annealing	Modified Best Fit (MBF)	O(τ×L×ln |R|)
SLPSO	Particle Swarm Optimization	The personal best (pbest)	O(M×K×log K)
EGA-ST	Genetic	blevel+tlevel	$O(\text{generations} \times n^2 \times p)$

Table 2
Definitions of notations.

Notation	Definition
T	A set of tasks in an application
E	A set of edges for precedence constraints among the tasks
P	A set of heterogeneous processors
g	Number of generations (iteration)
n	Number of tasks
p	Number of processors
e	Number of edges
m	The matrix size of Gaussian graph
T_i, t_i	The ith task in the application
P_k	The kth processor in the system
T_{exit}	The exit task with no successor
T_{entry}	The entry task with no predecessor
α	The parallelism factor
h	The range percentage of computation costs on processors
CP	The critical path
$PopSize$	The size of population
NL	The node level, representing the sum of tlevel and blevel of the node
LBP	Load balancing processor, representing the selected processor
PC	The processor counter, representing the number of processor
λ	The weight value for multi-objective fitness function
β	parallelization value

3. Task scheduling model

This section explains the task graphs of applications and mechanisms of task assignation to processors. In task graphs, every node or task is an instruction and every edge represents the dependence between two commands. Two nodes could be executed in parallel if they have no edge on task graphs. This paper uses two categories of real-world and random graphs to show the results of algorithms and compare them with other well-known algorithms of the field. Table 2 provides a list of variables and their definitions used in the paper.

3.1. Task graph generation metrics

The following parameters are employed to generate graphs with different characteristics to test algorithms.

- ***DAG size:*** This parameter determines the number of program tasks. In the task graph, *DAG size* is identified by a special parameter that is explained in Section 6.2.
- ***Computation cost of tasks:*** The amount of necessary computations for each task is shown by $W_d(T_i)$ and $S(T_i, P_k)$ is task execution speed T_i on P_k processor and task cost on a given processor is computed by Eq. (1) (Topcuoglu et al., 2002; Xu et al., 2014):

$$W(T_i, P_k) = \frac{W_d(T_i)}{S(T_i, P_k)}. \tag{1}$$

- ***Computation Cost Ratio (CCR):*** This parameter shows the average cost of communication divided by the average cost of task execution. If this value is very low, this graph shows a computation-intensive program. *CCR* value is computed by Eq. (2) (Barada et al., 2002; Xu et al., 2014):

$$CCR = \frac{\frac{1}{e}\sum_{edge(T_i,T_j)\in E} \overline{C(T_i, T_j)}}{\frac{1}{n}\sum_{T_i \in T} \overline{W(T_i)}}, \tag{2}$$

where $\overline{C(T_i, T_j)}$ is the average communication cost of the $edge(T_i, T_j)$ and $\overline{W(T_i)}$ is the average computation cost of T_i.

- ***Parallelism factor (α)*:** In random graph, graph depth is randomly generated with a uniform distribution that its average value is equal to $\sqrt{n}/\alpha$. Graph depth equal to the smallest value of greater integer is equal to the obtained decimal value. The number of nodes at each level is randomly created with uniform distribution and its average is equal to $\alpha\times\sqrt{n}$ if $\alpha\gg1$ we have maximum parallelism graph and if$\alpha\ll 1$ a minimum parallelism graph is generated (Topcuoglu et al., 2002).
- ***Computation cost heterogeneity factor (h)*:** The high value of h represents strong difference of computation cost for task execution on processors. A high h value indicates high variance of the computation costs of a task, with respect to the processors in the system. If h value is considered zero, it means that execution cost of each task is equal on all processors. Average cost of execution of each task T_i shown by $\overline{W(T_i)}$ is randomly achieved with a uniform distribution in the range of $[0,2\times\overline{w_{DAG}}]$. $\overline{w_{DAG}}$ is average graph runtime that is randomly considered not to affect comparing results. In this case, T_i runtime on P_k shown by $W(T_i, P_k)$ lies accidentally in Eq. (3) (Topcuoglu et al., 2002):

$$\overline{W(T_i)}\times\left(1-\frac{h}{2}\right)\quad\leq W(T_i, P_k)\leq\overline{W(T_i)}\times\left(1+\frac{h}{2}\right). \tag{3}$$

In some resources (Xu et al., 2014), h parameter along with Eq. (4) are determinants of processors symmetry level. For example, if h value is considered as half (0.5), asymmetry level equals 3 and if h value is 1, system will be symmetric.

$$\frac{1+h}{1-h},\ h\in(0,\ 1) \tag{4}$$

In our simulation, graphs were generated for all combinations of the above parameters. Every possible edge was created with the same probability, which was calculated based on the average number of edges per node. To obtain the required *CCR* for a graph, node weights are taken randomly from a uniform distribution. Edge weights are also taken from a uniform distribution, whose mean depends on *CCR* (0.1, 0.5, 1, 5 and 10) and Parallelism factor α (0.5). The relative deviation of the edge weights is the same to that of the node weights. Every set of the above parameters was used to generate several random graphs in order to avoid scattering effects. The results presented are the average of the results obtained for these graphs.

3.2. Scheduling model

In genetic algorithms, each scheduling case comes in form of a chromosome that includes assigned tasks and processors. In this method, each gene of the chromosome covers a pair of tasks and its given processor. This representation allows for easy management and by passing task prioritization phase of processor assignation.

To assign tasks to processors and to compute execution time, after creating schedules in EGA-TS, actual start time of task on processor is determined by Eq. (5) (Xu et al., 2014):

$$AST(T_i, P_k)=\max(EST(T_i, P_k),\ Avail(P_k)). \tag{5}$$

In this equation $Avail(P_k)$ shows the earliest time when P_k processor is ready to schedule, that is, previous task being executed on this processor has been completed.
$EST(T_i, P_k)$ is the earliest start time of a task on a single processor computed by Eq. (6) (Xu et al., 2014):

$$EST(T_i, P_k)=\begin{cases} 0, \quad if\ \ T_i=T_{entry}; \\ \max\limits_{Tj\in Pred(T_i)} AFT(T_j, P_l), \quad if\ \ P_k=P_l; \\ \max\limits_{Tj\in Pred(T_i)} (AFT(T_j, P_l)+C(T_j, T_i)), \quad if\ \ P_k\neq P_l. \end{cases} \tag{6}$$

In this equation $Pred(T_i)$ is the set of immediate predecessors of T_i and $C(T_j, T_i)$ is the communication cost between two tasks. $AFT(T_j, P_k)$ is the actual finish time of executing one task on a processor expressed by Eq. (7):

$$AFT(T_i, P_k)=AST(T_i, P_k)+W(T_i, P_k). \tag{7}$$

3.3. Fitness functions

Two fitness functions are proposed in EGA-TS. The single-objective function in the proposed algorithm is to minimize the *makespan*. The length of schedule resulted from algorithm *makespan* or scheduling fitness which is actual finish time (AFT) of exit task is defined by Eq. (8) (Xu et al., 2014). *Makespan* or completion time is the total time taken to process a set of tasks for its complete execution. Minimization of *makespan* can be done by assigning the set of tasks to set of processors or machines. After all task in a graph are scheduled, the schedule length (i.e., overall completion time) will be the actual finish time (AFT) of the exit task.

$$makespan=AFT(T_{exit}). \tag{8}$$

In most cases the results can be improved using multiple objective functions (Akbari and Rashidi, 2016; Alok et al., 2015). The multi-objective function (MOF) in the proposed algorithm is to minimize *makespan* and maximize parallelization in assigning the tasks to the processors. MOF is defined by Eq. (9).

$$MOF=\lambda\times\quad makespan+(1-\lambda)\times\ \frac{1}{\beta}\quad,\quad 0\leq\lambda\leq1 \tag{9}$$

where λ is the weight value randomly set by the proposed algorithm and β shows parallelization value which is defined by Eq. (10). Profit based task scheduling considers only maximization of parallelization and in this case λ=0. Likewise when λ=1, we emphasis on the minimization of the *makespan*. Our proposed algorithm makes a tradeoff between these two parameters and therefor more improved.

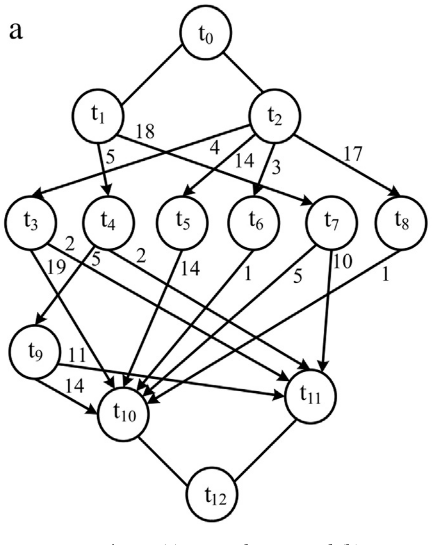
$$\beta=\frac{1}{\sum_{i=0}^{p-1}\left|\frac{n}{p}-|p_i|\right|}, \tag{10}$$

that the total number of tasks assigned to the processor p_i defined by $|p_i|$.

3.4. An example DAG *application*

Fig. 2 shows a task graph and task execution costs on 2 processors (Daoud and Kharma, 2011).

Fig. 3 shows some samples of schedules generated by the CPOP,



b

task	p0	p1
t0	0	0
t1	4	6
t2	15	22.5
t3	4	6
t4	13	19.5
t5	10	15
t6	7	10.5
t7	8	12
t8	4	6
t9	12	18
t10	6	9
t11	9	13.5
t12	0	0

Fig. 2. (a) A sample *DAG* and (b) Computation cost matrix.

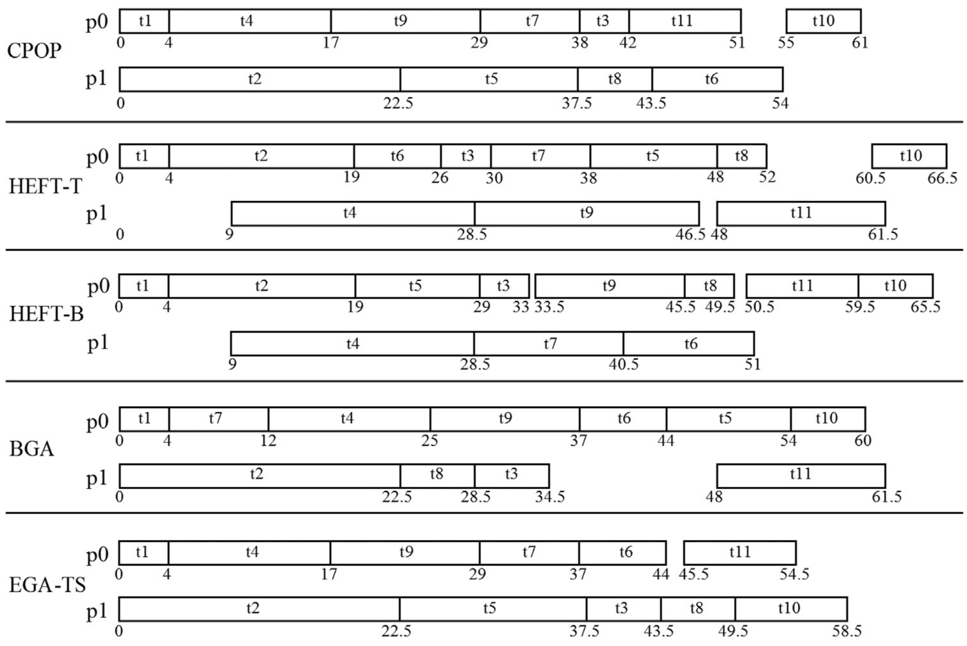


Fig. 3. The schedules generated by the CPOP, HEFT-T, HEFT-B, BGA and EGA-TS.

HEFT-T, HEFT-B, BGA (Gupta et al., 2010) and EGA-TS on Fig. 2 graph. The resulted fitness from EGA-TS equals 58.5 that is a better schedule compared to other algorithms. This result is achieved by 100 repetitions while in BGA algorithm, the result of 61.5 is obtained by 200 repetitions.

4. Algorithm design

A simple genetic algorithm includes three basic genetic operations: selection, crossover, and mutation. In selection, solutions from the population are selected as parents; in crossover, the parents are crossbred to produce child; and in mutation, the offspring may be altered according to mutation rules. In genetic algorithms, iterations of an algorithm is called generations.

For task scheduling, genetic algorithms assigns tasks to processors in a random way at first. These schedules generate a population. The best are selected and duplicated by using a cross-over operator and some are mutated given their environmental conditions to create a new generation. This is repeated enough to obtain the best scheduling or solution on an evolution-based path (Gupta et al., 2010). However, in EGA-TS, the initial populations are not purely identical and possess relatively optimized schedules. Moreover, operators have been improved and interspersed with new ones. The pseudocode of EGA-TS is shown in Algorithm 1.

Algorithm 1. Pseudocode of EGA-TS.

```
Inputs:
  A DAG application,
  The Crossover Probability,
  The Mutation Probability,
  The Size of Run,
  The Size of Population,
  The Number of Processors;
Output:
  A task schedule;
1: call InitialPopulation;
2: repeat
3: call Selection;
4: call Crossover;
5: call Mutation;
6: call Inversion;
7: until Size of Run;
```

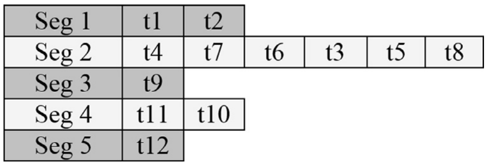
4.1. Initial populations

Random treatment of initial population selection leaves us with unsuitable search space. Therefore we should generate initial populations that allow us to reach relatively optimized solutions in reasonable runtime. This is possible through application of heuristic model which is shown in Algorithm 2.

Algorithm 2. Initial Population.

```
Input:
  A DAG application;
Output:
  The initial population;
1: call Segmentation;
2: repeat
3:    foreach segment generated by Segmentation do
4:    foreach nodes in a segment do
5:    if $NL==|CP|$
6:    Assign it to the fastest processor;
7:    else
8: Assign it to the LBP;
9:    endif
10:    endfor
11: endfor
12: until Size of Population;
```

This algorithm uses segmentation function to divide scheduling into several segments. Each segment should cover independent tasks that allow for parallel execution on different processors. This could be followed by relocating tasks of a given segment for generation of more genetically diversified chromosomes. Fig. 4 presents four segments of the graph in Fig. 2. After executing the segmenting function, tasks on critical paths are selected and then prepared for the fastest processor to execute them. A critical path is a path from entry to exit nodes with highest computation and communication costs of edges and also



Seg 1	t1	t2				
Seg 2	t4	t7	t6	t3	t5	t8
Seg 3	t9					
Seg 4	t11	t10				
Seg 5	t12					

Fig. 4. Segmentation of graph in Fig. 2.

influence on the whole scheduling. Load balancing is the final part of operation. In this part, tasks are distributed in balanced way among processors, i.e. each segment uses all the processors to prevent idleness.

NL (the Node level) and $|CP|$ (the value of Critical Path) are calculated by Eqs. (11) and (12) which come as below:

$$NL = tlevel(t_i) + blevel(t_i) \text{ and} \tag{11}$$

$$|CP| = (tlevel(T_{entry}) + blevel(T_{entry})) \ or \ (tlevel(T_{exit}) + blevel(T_{exit})), \tag{12}$$

where *tlevel* and *blevel* are identified by Eqs. (13) and (14):

$$tlevel(t_i) = \max_{t_j \in imed_{pred}(t_i)} \{tlevel(t_j) + \overline{w_j} + \overline{c_{j,i}}\}, \tag{13}$$

$$blevel(t_i) = \overline{w_i} + \max_{t_j \in imed_{succ}(t_i)} \{blevel(t_j) + \overline{c_{i,j}}\}. \tag{14}$$

The *LBP* is load balancing processor which is defined by Eq. (15):

$$LBP = PC mod p, \tag{15}$$

in this equation *PC* is the processor counter.

4.2. Selection function

Genetic algorithms use a variety of selection functions, including Rank Selection, Steady-State Selection, Elitism and Roulette-wheel selections. Some researches show that Roulette-wheel method is an ideal option for implementation of selection operators (Ahmad et al., 2016; Farrag et al., 2015; Kalra and Singh, 2015; Kumar and Vidyarthi, 2016). This method enhances the chances of fitter chromosomes for selection. All solutions are placed on roulette wheel where better solution has larger portion on the wheel. This gives a fair chance to each solution to be a potential parent in proportion to their fitness value. It can be described as follows. Some solutions will be selected to undergo genetic operations for reproduction according to their fitness values. A chromosome having a higher fitness value should therefore have a higher chance to be selected. It works by calculating the selection probability ($prob_i$) of each chromosome using Eq. (16) (Xu et al., 2014). Then sum_i which is the sum if probabilities from 1 to i is calculated by Eq. (17) (Xu et al., 2014) and is carried over to next generation through application of Algorithm 3. In this algorithm, 10% of the best chromosomes are duplicated in the next generation and the rest will be transmitted once marked by the selection function.

$$Prob_i = \frac{fitness_i}{\sum_{j=1}^{PopSize} fitness_j} \tag{16}$$

$$Sum_i = \sum_{j=1}^{i} Prob_j \tag{17}$$

Algorithm 3. Selection.

```
Input:
  Current population;
Output:
  New Selected Population;
1: Select 10% of best chromosomes base on makespan and copy to
   the Selected Population;
2: Generate a random number $R \in [0, 1]$;
3: repeat
4:    if $Sum_i > R$ then
5:    Select the iTh chromosome and add to the Selected Population;
6:    endif
7: until Size of Population;
```

4.3. Cross-over function

Cross-over functions are used for generation of new chromosomes if they can fulfill the following conditions (Xu et al., 2014):

- **Correctness:** following the prioritized dependency of tasks.
- **Completeness and Uniqueness:** non-repeated occurrence of all tasks in a given chromosome.

Theorem 1. A solution of task scheduling is an execution order which is a topological order of tasks based on their dependencies. It is still a topological order without violating precedence constraints when the task T_i is removed from the topological order (i.e., the priority queue).

Proof 1. When the task T_i is removed from a topological order, the remaining priority queue is actually a topological order of the new DAG obtained by removing T_i from the original DAG. Hence, all precedence constraints of the new DAG are preserved in the remaining priority queue, which is certainly a topological order of the new DAG.

Theorem 2. The task T_i can be inserted into any position between $Pred(T_i)$ and $Succ(T_i)$, which can provide a new topological order (i.e., priority queue) without violating precedence constraints.

Proof 2. We know that all tasks between $Pred(T_i)$ and $Succ(T_i)$ are independent of T_i. In other words, if the task T_j is between $Pred(T_i)$ and $cc(T_i)$, then there is no precedence constraint between T_i and T_j. Hence, the relative order between T_i and T_j in any topological order can be arbitrary. This implies that T_i can be inserted into any position between $Pred(T_i)$ and $Succ(T_i)$.

Two chromosomes are randomly selected as parents with probability P_c and 2 children are generated under above-mentioned conditions by following Fig. 5. This algorithm uses a single-point cross-over which is used for random selection of parents. We have marked the cross-over as in Fig. 5. For the first child, the initial selected

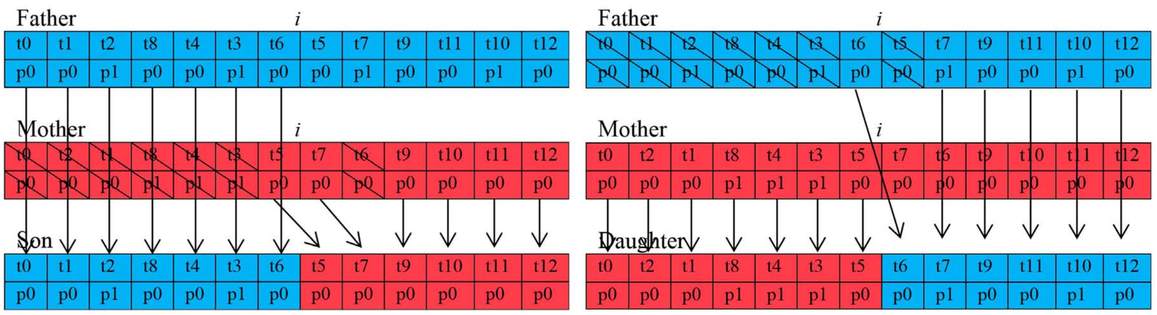
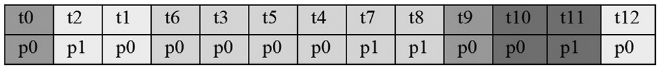

Father
i
t0 t1 t2 t8 t4 t3 t6 t5 t7 t9 t11 t10 t12
p0 p0 p1 p0 p0 p1 p0 p0 p1 p0 p0 p1 p0
Mother
i
t0 t2 t1 t8 t4 t3 t5 t7 t6 t9 t10 t11 t12
p0 p0 p0 p1 p1 p1 p0 p0 p0 p0 p0 p0 p0
Son
t0 t1 t2 t8 t4 t3 t6 t5 t7 t9 t10 t11 t12
p0 p0 p1 p0 p0 p1 p0 p0 p0 p0 p0 p0 p0
Daughter
t0 t2 t1 t8 t4 t3 t5 t6 t7 t9 t11 t10 t12
p0 p0 p0 p1 p1 p1 p0 p0 p1 p0 p0 p1 p0

Fig. 5. Cross-over operation on 2 selected chromosomes.



t0	t2	t1	t6	t3	t5	t4	t7	t8	t9	t10	t11	t12
p0	p1	p0	p0	p0	p0	p0	p1	p1	p0	p0	p1	p0

Fig. 6. Mutation areas in chromosomes of the graph of Fig. 2.

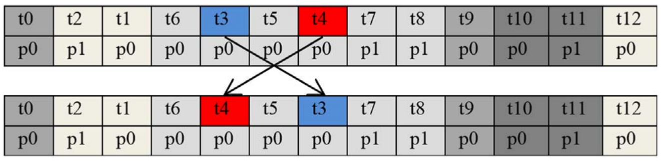

t0 t2 t1 t6 t3 t5 t4 t7 t8 t9 t10 t11 t12
p0 p1 p0 p0 p0 p0 p0 p1 p1 p0 p0 p1 p0
t0 t2 t1 t6 t4 t5 t3 t7 t8 t9 t10 t11 t12
p0 p1 p0 p0 p0 p0 p0 p1 p1 p0 p0 p1 p0

Fig. 7. Mutation operator.

chromosomes of the father are directly transmitted to the child. The same goes about the mother's chromosomes. This proves the correctness, completeness and uniqueness of chromosomes (according to Theorems 1 and 2 (Xu et al., 2014)). The process is almost the same for the second child but different as the order of transmission is revered. First mother's chromosomes and then father's ones will be transmitted. These stages are depicted in Algorithm 4.

4.4. Mutation function

This function alters genes to develop searching space with P_m probability. This leads to increased variety of samples to avoid local optimums. These changes should not affect the prioritization of genes and therefore changes of each segment must take place by segmentation function in given segments which are called *mutation areas* (Fig. 6). A t_i gene is randomly selected in each segment and is transposed with a gene of the same segment. Fig. 7 shows a mutation operation of a chromosome. Details are given in Algorithm 5.

Theorem 3. In *mutation areas*, tasks can be displaced in any of schedules without dependences being violated in task graph.

Proof 3. Existing tasks in a segment have the capacity of parallel execution. That is, their order can be changed.

According to the Theorem 3, the mutation function proves the correctness of chromosomes.

4.5. Inversion function

In addition to standard genetic operators, a new operator or inversion function inverses the order of available genes marked by the segmentation function. These changes guarantee the correctness of chromosomes due to independence of tasks in each segment (according to Thorem3). Fig. 8 shows an inversion operation performed on a chromosome. The details are given in Algorithm 6.

Algorithm 4. Crossover.

```
Inputs:
   Two parents from the current population.
Outputs:
   Two new children.
1: Choose randomly crossover point $i$;
2: Cut the father's chromosome and the mother's chromosome from
   $i$;
3: Copy genes in first part of father's chromosome to the son's
   chromosome;
4: Inherit the genes of the mother's chromosome that do not appear
   in the left part of father's chromosome to the right part of the
   son's chromosome;
5: Copy genes in first part of mother's chromosome to daughter's
   chromosome;
6: Inherit the genes of the father's chromosome that do not appear
   in the left part of mother's chromosome to the right part of the
   daughter's chromosome;
7: Replace these two new children with two random selected
   chromosomes in population.
```

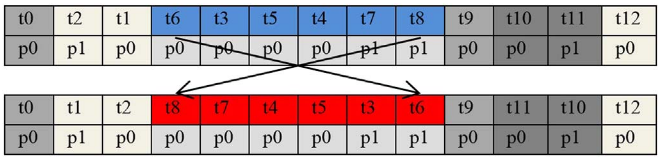

t0 t2 t1 t6 t3 t5 t4 t7 t8 t9 t10 t11 t12
p0 p1 p0 p0 p0 p0 p0 p1 p1 p0 p0 p1 p0
t0 t1 t2 t8 t7 t4 t5 t3 t6 t9 t11 t10 t12
p0 p1 p0 p0 p0 p0 p0 p1 p1 p0 p0 p1 p0

Fig. 8. Inversion operation on the chromosome of Fig. 6.

Algorithm 5. Mutation.

```
Input:
   A randomly chosen chromosome.
Output:
   A new chromosome.
1: Choose randomly a gene $t_i$;
2: Generate a new child by interchanging the task of gene $t_i$ with a
task of a gene in the same segment;
```

5. Time and space complexity evaluation

The time complexity of EGA-TS is evaluated as following: According to Algorithm 1, the time is mostly spent in running the loop (Steps 2–7) in the EGA-TS. In each iteration of the loop, the algorithm needs to execute Selection function, Crossover function, Mutation function and Inversion function. The time complexity of the Selection function is $O(n^2)$. The time complexity of the Mutation function is $O(n^2)$. The time complexity of the Inversion function is $O(n^2)$. The time complexity of the fitness evaluation function is $O(e\times p)$, where e is the number of edges in the *DAG* and p is the number of processors. Therefore, the time complexity of EGA-TS is $O(g\times(n^2+n^2+n^2+e\times p))$, where g is the number of generation performed by EGA-TS. For a dense graph where the number of edges is $O(n^2)$, the time complexity is $O(g\times n^2\times p)$. In compile time scheduling, the number of generation for achieving an appropriate solution does not affect execution time required for next times since the program is compiled once and run for several times.

The space complexity of EGA-TS is analyzed as follows: In EGA-TS, to store each schedule, an array of size $n\times 2$ is needed. There are *PopSize* schedules in the initial population, hence, the space complexity of EGA-TS is $O(PopSize\times n\times 2)$.

Algorithm 6. Inversion.

```
Input:
   Current population.
Output:
   A new inverted population.
1: foreach chromosome in population do
2:    foreach level in current chromosome do
3:    Reverse the order of tasks;
4:    endfor;
5: endfor;
```

6. Performance analysis and discussion

To show the efficiency of EGA-TS for task scheduling in heterogeneous systems, results of running this algorithm are compared with HEFT-T, HEFT-B, CPOP (Topcuoglu et al., 2002), BGA (Gupta et al., 2010), SA (Damodaran and Vélez-Gallego, 2012) and SLPSO (Zuo et al., 2014) algorithms. These comparisons using the criteria outlined in Section 6.1.

EGA-TS is implemented in C# language and it consists of different classes to create standard graphs including Fast Fourier Transformation (FFT), Molecular, Gaussian graphs and random graph with various parameters. To maintain one population schedules, a matrix equal to the number of desired population where each row of this matrix equals to the number of existing tasks in schedule is applied. The simulations are performed on the same PC with an Intel processor Core i7@2.2 GHz and 6 GB RAM.

6.1. Performance comparison metrics

The following metrics are usually employed to compare performance of scheduling algorithms on standard graphs.

- ***Scheduling Length Ratio (SLR)*:** The most important performance measurement criterion on graph scheduling algorithms is scheduling length resulted from output algorithm. Since a large number of graphs with different characteristics are used, it is necessary to normalize scheduling length to a low band for every graph so that we reach a criterion for total comparison called scheduling length ratio that is expressed by Eq. (18) (Burkimsher et al., 2013; Dai and Zhang, 2014; Ijaz et al., 2013; Kumar and Katti, 2014; Neubert et al., 2010; Savino et al., 2015; Topcuoglu et al., 2002; Xu et al., 2014; Zhang et al., 2014):

$$SLR=\frac{makespan}{\sum_{T_i\in CP_{min}} min_{P_k\in P}(W(T_i,P_k))}. \tag{18}$$

In this equation, CP shows critical path and $W(T_i,P_k)$ indicates execution time of T_i task on P_k processor. In a non-scheduled graph, if calculation cost for each task is considered as the lowest execution time on processors, critical path will be computed based on the lowest execution cost indicated by CP_{min}. In fact, this criterion divides resulted time into best schedule (CP_{min} task execution sum on the fastest processor) to obtain a normal criterion. SLR average on several graphs is considered as comparison criterion. Scheduling length ratio cannot be less than 1. In some sources (Bansal et al., 2003; Daoud and Kharma, 2011), this metric is known as Normalized Scheduling Length (NSL) and is computed by Eq. (19):

$$NSL=\frac{\text{Schedule Length}}{\sum_{T_i\in CP_{min}} C_{i,a}}, \tag{19}$$

where $C_{i,a}$ is the cost of transmission from T_i to T_a when executed on various processors.

- ***Speedup parameter*:** This criterion is resulted from dividing speed of task continous execution on fastest processor in parallel execution and is expressed by Eq. (20) (Burkimsher et al., 2013; Dai and Zhang, 2014; Ijaz et al., 2013; Kumar and Katti, 2014; Topcuoglu et al., 2002; Xu et al., 2014; Zhang et al., 2014):

$$Speedup=\frac{\boldsymbol{min}_{P_k\in P}(\sum_{T_i\in T} W(T_i,P_k))}{makespan}. \tag{20}$$

- ***Efficiency metric*:** Efficiency is another criterion that shows *speedup* ratio to the number of applied processor.

6.2. Real-world application graphs

In this paper, three standard task graphs are utilized as real-world application graphs to evaluate the algorithms. These graphs include Molecular Dynamic Code, Gaussian elimination, and FFT graphs (Daoud and Kharma, 2011; Mohamed and Awadalla, 2011; Topcuoglu et al., 2002; Xu et al., 2014). All of these graphs along with their generation algorithms will be explained in this section.

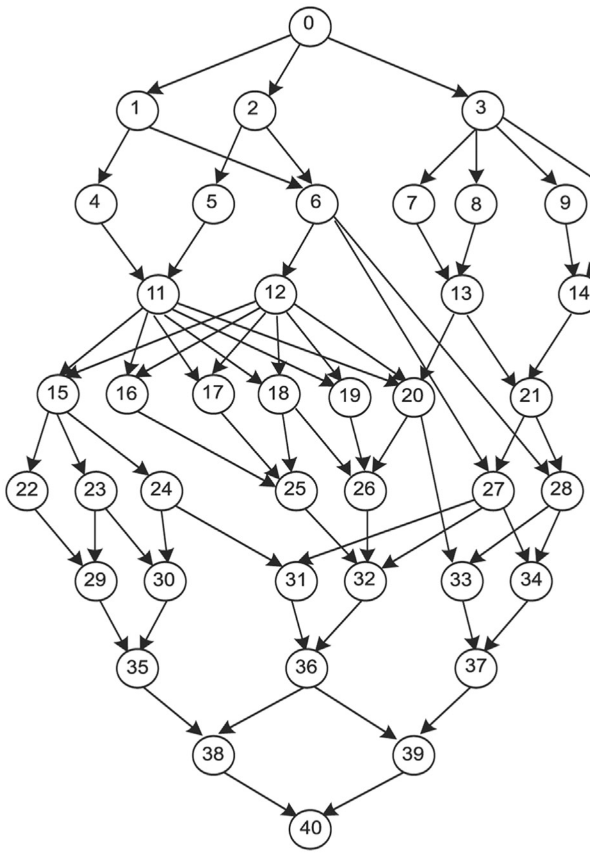


Fig. 9. Sample of molecular graph.

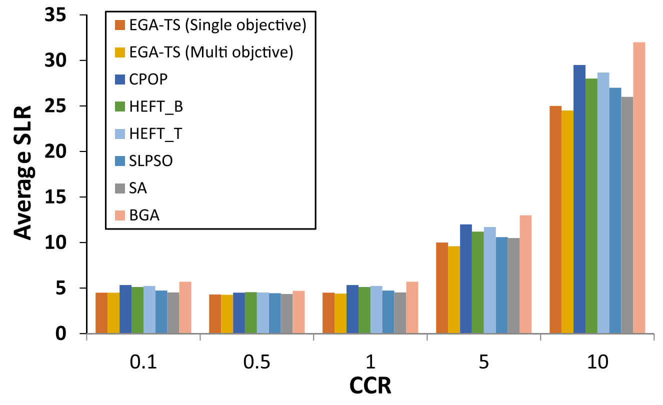
6.2.1. Molecular graph

A sample of Molecular graph is given in Fig. 9 (Kim and Browne, 1988). This graph is used as a graph with a fixed value to compare scheduling algorithms.

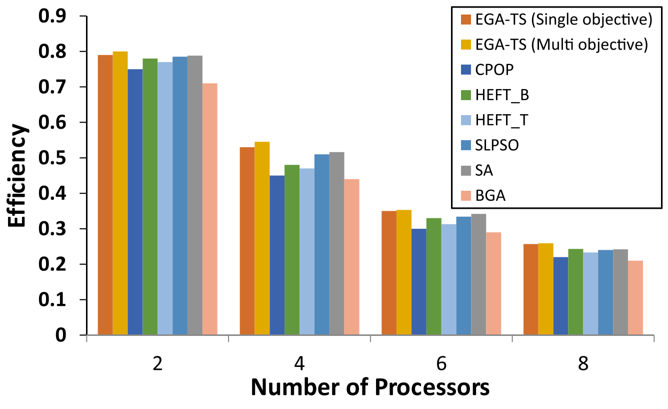
Metrics to create 100 molecular graphs and BGA and EGA-TS input parameters are given in Table 3. In Fig. 10(a), SLR average of scheduling algorithms versus different CCR values are presented that express EGA-TS superiority over other algorithms. The findings of this research demonstrated that increased CCR improves the performance of the algorithm. This indicates that EGA-TS still performs better in spite of increased communication costs. In Fig. 10(b), the efficiency of algorithms versus different number of processors is evaluated showing EGA-TS superiority over other algorithms. These results show that EGA-TS will still perform better than other algorithms if more processors are considered, while BGA as a metaheuristic algorithm has a poor performance compared with other algorithms. This is because BGA requires more repetitions to produce the proper solution while EGA-TS has a higher efficiency with the same number of repetitions.

Table 3
Molecular graph generation metrics for different parameters of graph scheduling algorithm inputs.

DAG					
Type	Number of *DAG*s	α	h	CCR	Node number
Molecular	100	0.5	0.1	1	40
Metaheuristic parameters					
P_m (Mutation probability)	P_c (Cross over probability)	Population Number		Generation Number	
Random select from [0.05,1]	Random select from [0.05,1]	30		100	



(a) Average SLR of the algorithms for different values of CCR



(b) Efficiency of the algorithms based on the number of processors

Fig. 10. Average SLR and the efficiency of algorithms for molecular graph.

6.2.2. *Gaussian graph*

Gaussian graph results from using Fig. 11(a) algorithm and identifying value of matrix m. In this algorithm, $T_{k,k}$ are main nodes expressing main operations and $T_{k,j}$ is considered as their dependent nodes in graph. Total number of nodes in this graph is $(m^2+m-2)/2$ (Topcuoglu et al., 2002). Communication cost for edges and calculation cost in each level for nodes are computed by a normal distribution with values of two *CCR* and h metrics. Fig. 11(b) shows a Gaussian graph sample with matrix sized of five (Topcuoglu et al., 2002).

The results obtained from algorithm execution on 100 Gaussian graphs with the parameters given in Table 4 and various input points from 5 to 20 suggesting superiority of EGA-TS over others are presented in Fig. 12. The finding of this section shows that a bigger *DAG size* denotes its complexity. Moreover, EGA-TS still produces shorter-length schedules in spite of lower repetitions (50), i.e. it distributes program tasks among processors in a way to minimize the total execution time of the program. These results also show that the use of multi-objective fitness function improved the scheduling.

Table 4
Gaussian graph generation metrics for different graph scheduling algorithm inputs parameters.

DAG					
Type	Number of *DAGs*	α	h	*CCR*	Processor Number
Gaussian	100	0.5	0.1	1	8
Metaheuristic parameters					
P_m (Mutation probability)	P_c (Cross over probability)	Population Number		Generation Number	
Random select from [0.05,1]	Random select from [0.05,1]	40		50	

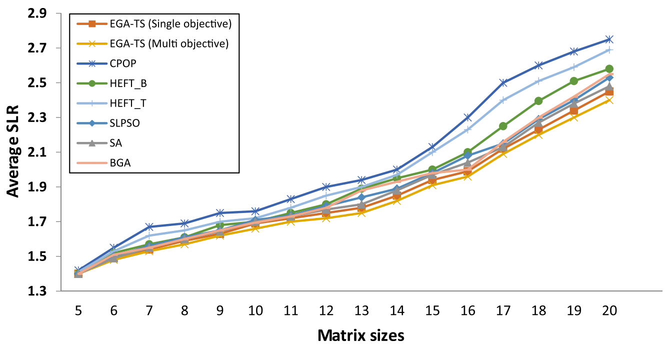
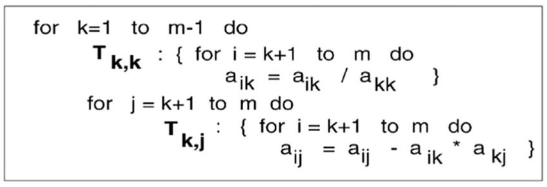


Fig. 12. Average SLR of algorithms for different the matrix sizes of Gaussian graph.

6.2.3. *FFT graph*

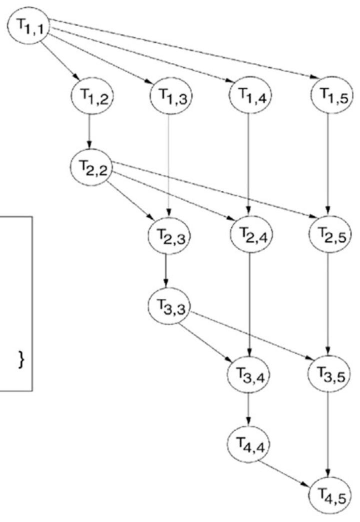
FFT graph is generated by an algorithm given in Fig. 13(a). In this algorithm, A is an array of size n for maintaining polynomial coefficients and Y is an array showing algorithm output (Topcuoglu et al., 2002).

This algorithm consists of two parts: in lines three and four, there is a recursive call that creates the generated graph above dotted line in



```
for  k=1  to  m-1  do
        T_k,k  : { for  i = k+1  to  m  do
                        a_ik  =  a_ik  /  a_kk     }
     for  j = k+1  to  m  do
             T_k,j  :  { for  i = k+1  to  m  do
                            a_ij  =  a_ij  -  a_ik  *  a_kj   }
```

(a)

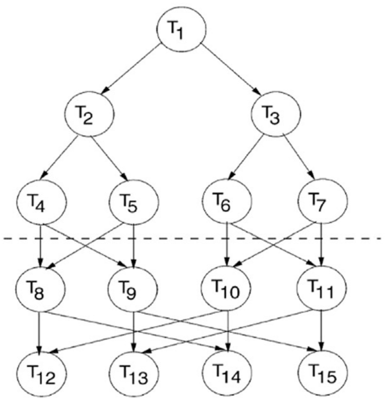


(b)

Fig. 11. (a) Gaussian graph generation algorithm. (b) Gaussian graph sample with matrix sized of five.

```
FFT (A, ω)
1.    n = length(A)
2.    if  (n=1)  return (A)
3.    Y^(0) = FFT( (A[0],A[2],...A[n-2]) , ω^2 )
4.    Y^(1) = FFT( (A[1],A[3],...A[n-1]) , ω^2 )
5.    for  i = 0  to  n / 2 -1  do
6.        { Y[i] = Y^(0)[i] + ω^i * Y^(1)[i]
7.          Y[i+n/2] = Y^(0)[i] - ω^i * Y^(1)[i] }
8.    return (Y)
```

(a)

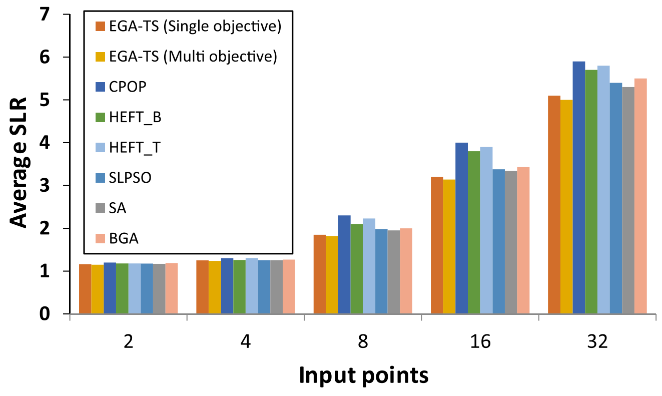


(b)

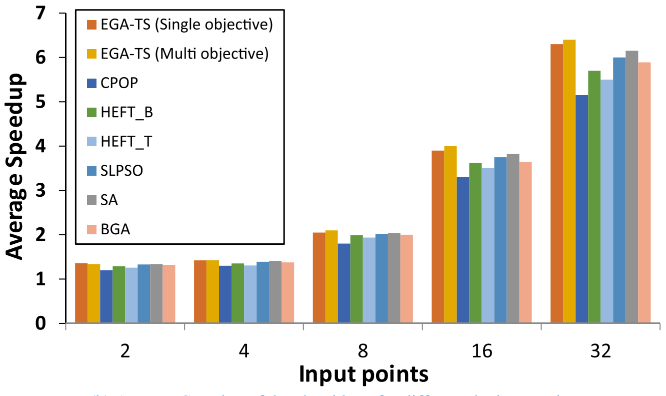
Fig. 13. (a) FFT algorithm. (b) A sample generated graph by this algorithm.

Fig. 13(b) (recursive nodes) and in lines six and seven there is a butterfly operation that creates nodes below dotted line. For a vector of size n, return nodes number equals 2×n-1 and butterfly operation recursive nodes is $n \times log_2^n$ (n size is considered as a power of 2). In generated graphs by this algorithm, all paths from start node to exit node are critical paths (Topcuoglu et al., 2002). To generate this graph, *CCR* and *h* parameters elaborated in graph generation parameters section together with *input points* indicating the number of graph leaves are utilized. Fig. 13(b) shows a sample FFT graph with the number of input points of 4 (4 is the number of graph leaves).

Fig. 14(a) and (b) give 100 graph comparison average of *SLR* and *speedup* with different input points from 2 to 32 showing EGA-TS (Multi-objective) superiority over EGA-TS (Single-objective) and other algorithms. Although the small graphs do not display high scheduling differences, creation of schedules by EGA-TS's improves more than other parameters once *DAG* increases in size Parameters of graph generation and scheduling algorithms inputs are presented in Table 5.



(a) Average SLR of the algorithms for different the input points



(b) Average Speedup of the algorithms for different the input points

Fig. 14. Average SLR and the Speedup of algorithms for FFT graph.

Table 5
FFT graph generation metrics for different parameters of graph scheduling algorithm input.

DAG					
Type	Number of *DAGs*	*α*	*h*	*CCR*	Processor Number
FFT	100	0.5	0.1	1	4
Metaheuristic parameters					
P_m (Mutation probability)		P_c (Cross over probability)	Population Number		Generation Number
Random select from [0.05,1]		Random select from [0.05,1]	40		50

6.3. Random graph

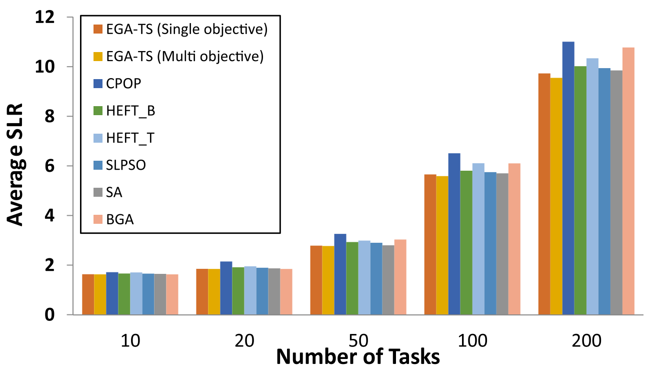
Several random graphs are generated by random graph generation software. In a random graph, the number of nodes (*DAG* size) and the number of edges of each node is received stably from SET_v and SET_{out_degre} sets and graph depth is randomly created by a uniform distribution where its average value equals $\sqrt{n}/\alpha$ and the number of nodes in each level is also randomly created by a uniform distribution where its average value equals $\alpha \times \sqrt{n}$. In random graph, communication costs are assumed equal for all nodes (Topcuoglu et al., 2002).

SLR and *speedup* average comparisons of 100 random graphs with different sized from 10 to 200 are given in Fig. 15(a) and (b) showing better performance of EGA-TS compared to others. The findings of this evaluation show that *speedup* parameters and *SLR* improve in the presence of larger *DAG*. In other words, EGA-TS performs better for bigger complex randomized graphs in comparison to other algorithms and multi-objective fitness function superiority over single-objective fitness function. The parameters for graph generation and scheduling algorithms inputs are presented in Table 6.

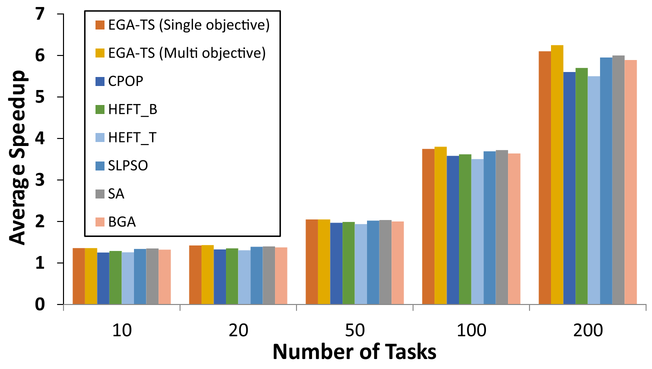
6.3.1. Convergence trace

Convergence trace is utilized to compare EGA-TS repletion number with BGA algorithm. This trace shows which algorithm reaches a better schedule in lower number of repetitions. Results of this trace in Fig. 16(a) and (b) carried out respectively on graphs of sizes 10 and 20 shows that EGA-TS reaches a better schedule in a lower number of repetitions compared to BGA algorithm. Graph generation metrics versus algorithm inputs are shown in Table 7.

Although EGA-TS needs lower repetitions than their genetic counterparts, it needs high frequency of repetition to produce the desired answer. This is a drawback for this algorithm compared to



(a) Average SLR of the algorithms for different the number of tasks



(b) Average Speedup of the algorithms for different the number of tasks

Fig. 15. Average SLR and the speedup of algorithms for random graph.

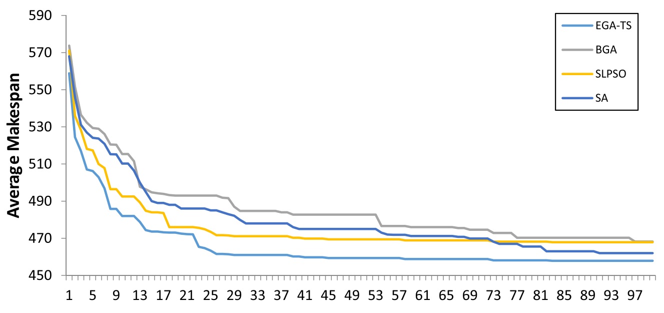
Table 6
Random graph generation metrics for different scheduling algorithm input.

DAG					
Type	Number of *DAGs*	α	*h*	*CCR*	Processor Number
Random	100	0.5	0.1	1	3
Metaheuristic parameters					
P_m (Mutation probability)	P_c (Cross over probability)		Population Number		Generation Number
Random select from [0.05,1]	Random select from [0.05,1]		30		25

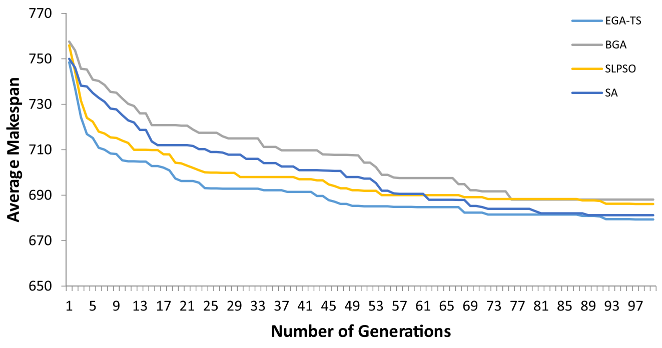
heuristic algorithms such as CPOP and HEFT. Furthermore, the volumes of entry-level parameters of the algorithm such as P_c *and* P_m which affect the outcomes are set manually without any involvement of the algorithm. Generation of initial population by other heuristic methods will be studied in future works and also a self-adaptive mechanism will be added to the proposed mechanism in future using self-adaptive operators. It can change algorithm input parameters such as cross-over probability and mutation by each repetition and move toward optimization. Finally, we will used new approaches to increase tasks to processor allocation performance.

7. Conclusions

This paper presented some significant changes to improve efficiency of the basic genetic algorithms for scheduling tasks in heterogeneous systems. Our algorithm, EGA-TS, generates an initial population embedded with non-random schedules that is in contrast to basic genetic algorithms. The function of initial population generation creates schedules with relatively optimized execution time. Also, genetic functions are improved to guarantee variety and consistent coverage of problem space. While generating schedules with lowered repetitions that allow for a maximum of parallelization for performance



(a) The convergence of makespan for randomly generated DAGs with 10 tasks, 50 independent runs



(b) The convergence of makespan for randomly generated DAGs with 20 tasks, 50 independent runs

Fig. 16. Convergence trace for random graph.

Table 7
Random graph generation metrics for different graph scheduling inputs of convergence trace.

DAG					
Out degree	Number of tasks	α	*h*	*CCR*	Processor Number
5	10, 20	0.5	0.1	1	6
Metaheuristic parameters					
P_m (Mutation probability)	P_c (Cross over probability)		Population Number		Generation Number
Random select from [0.05,1]	Random select from [0.05,1]		40		100

of the tasks, it reduces execution time and helps to avoid conflicts with local optimums. The results of this algorithm have been compared with the algorithms HEFT-T, HEFT-B, CPOP, BGA, SA and SLPSO based on standard parameters. The results reveal superior performance of EGA-TS over HEFT-T, HEFT-B and CPOP. Also, it offers better scheduling with lowered repetitions in comparison with BAG, SA and SLPSO as metaheuristic approaches. The use of multi-objective fitness function better results than single-objective fitness function. Multi-objective fitness function by increasing concurrency reduce the total execution time and improves scheduling efficiency.

References

Ahmad, S.G., Liew, C.S., Munir, E.U., Ang, T.F., Khan, S.U., 2016. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. J. Parallel Distrib. Comput. 87, 80–90.
Akbari, M., Rashidi, H., 2016. A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems. Expert Syst. Appl. 60, 234–248.
Alok, A.K., Saha, S., Ekbal, A., 2015. A new semi-supervised clustering technique using multi-objective optimization. Appl. Intell. 43, 633–661.
Babukartik, R., Dhavachelvan, P., 2012. Hybrid algorithm using the advantage of ACO and Cuckoo search for job scheduling. Int. J. Inf. Technol. Converg. Serv., 2.
Bandyopadhyay, S., Saha, S., Maulik, U., Deb, K., 2008. A simulated annealing-based multiobjective optimization algorithm: AMOSA. IEEE Trans. Evolut. Comput. 12, 269–283.
Bansal, S., Kumar, P., Singh, K., 2003. An improved duplication strategy for scheduling

precedence constrained graphs in multiprocessor systems. Parallel Distrib. Syst. IEEE Trans. on 14, 533–544.
Barada, H., Sait, S.M., Baig, N., 2002. A simulated evolution approach to task matching and scheduling in heterogeneous computing environments. Eng. Appl. Artif. Intell. 15, 491–500.
Burkimsher, A., Bate, I., Indrusiak, L.S., 2013. A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times. Future Gener. Comput. Syst. 29, 2009–2025.
Dai, Y., Zhang, X., 2014. A synthesized heuristic task scheduling algorithm. Sci. World J., 2014.
Damodaran, P., Vélez-Gallego, M.C., 2012. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. Expert Syst. Appl. 39, 1451–1458.
Daoud, M.I., Kharma, N., 2011. A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. J. Parallel Distrib. Comput. 71, 1518–1531.
Farrag, A.A.S., Mahmoud, S.A., El-Horbaty, E., 2015. Intelligent cloud algorithms for load balancing problems: a survey. 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE, pp. 210–216.
Ferrandi, F., Lanzi, P.L., Pilato, C., Sciuto, D., Tumeo, A., 2010. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. on 29, 911–924.
Gogos, C., Valouxis, C., Alefragis, P., Goulas, G., Voros, N., Housos, E., 2016. Scheduling independent tasks on heterogeneous processors using heuristics and column pricing. Future Gener. Comput. Syst. 60, 48–66.
Guo, L., Zhao, S., Shen, S., Jiang, C., 2012. Task scheduling optimization in cloud computing based on heuristic algorithm. J. Netw. 7, 547–553.
Gupta, S., Agarwal, G., Kumar, V., 2010. Task scheduling in multiprocessor system using genetic algorithm, Machine Learning and Computing (ICMLC). 2010 Second International Conference on. IEEE, pp. 267–271.
Hwang, R., Gen, M., Katayama, H., 2006. A performance evaluation of multiprocessor scheduling with genetic algorithm. Asia Pac. Manag. Rev. 11, 67.
Ijaz, S., Munir, E.U., Anwar, W., Nasir, W., 2013. Efficient scheduling strategy for task graphs in heterogeneous computing environment. Int. Arab J. Inf. Technol. 10, 486–492.
Kalra, M., Singh, S., 2015. A review of metaheuristic scheduling techniques in cloud computing. Egypt. Inform. J. 16, 275–295.
Kim, H., Kang, S., 2011. Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using Ant Colony Optimization. Inf. Sci. 181, 3995–4008.
Kim, S., Browne, J., 1988. A general approach to mapping of parallel computation upon multiprocessor architectures. International Conference on Parallel Processing, p. 8.
Kołodziej, J., Khan, S.U., 2012. Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. Inf. Sci. 214, 1–19.
Kumar, N., Vidyarthi, D.P., 2016. A novel hybrid PSO–GA meta-heuristic for scheduling of DAG with communication on multiprocessor systems. Eng. Comput. 32, 35–47.
Kumar, V., Katti, C., 2014. A scheduling approach with processor and network heterogeneity for grid environment. Int. J..
Kwok, Y.-K., Ahmad, I., 1996. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. Parallel Distrib. Syst. IEEE Trans. on 7, 506–521.
Lin, C.-S., Lin, C.-S., Lin, Y.-S., Hsiung, P.-A., Shih, C., 2013. Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems. J. Syst. Archit. 59, 1083–1094.
Lin, J., Zhong, Y., Lin, X., Lin, H., Zeng, Q., 2014. Hybrid Ant Colony Algorithm Clonal Selection in the Application of the Cloud's Resource Scheduling. arXiv preprint arXiv:1411.2528.
Lo, S.-T., Chen, R.-M., Huang, Y.-M., Wu, C.-L., 2008. Multiprocessor system scheduling with precedence and resource constraints using an enhanced ant colony system. Expert Syst. Appl. 34, 2071–2081.
Lu, H., Niu, R., Liu, J., Zhu, Z., 2013. A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem. Appl. Soft Comput. 13, 2790–2802.
Manudhane, K.A., Wadhe, A., 2013. Comparative study of static task scheduling algorithms for heterogeneous systems. Int. J. Comput. Sci. Eng., 5.
Mishra, P.K., Mishra, A., Mishra, K.S., Tripathi, A.K., 2012. Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules. Appl. Math. Model. 36, 6243–6263.
Mohamed, M.R., Awadalla, M.H., 2011. Hybrid algorithm for multiprocessor task scheduling. Int. J. Comput. Sci. Issues 8, 79–89.
Navimipour, N.J., Milani, F.S., 2015. Task scheduling in the cloud computing based on the cuckoo search algorithm. Int. J. Model. Optim. 5, 44.
Neubert, G., Savino, M.M., Pedicini, C., 2010. Simulation approach to optimize production costs through value stream mapping. Int. J. Oper. Quant. Manag. 16, 1–21.
Omara, F.A., Arafa, M.M., 2010. Genetic algorithms for task scheduling problem. J. Parallel Distrib. Comput. 70, 13–22.
Pendharkar, P.C., 2011. A multi-agent memetic algorithm approach for distributed object allocation. J. Comput. Sci. 2, 353–364.
Pendharkar, P.C., 2015. An ant colony optimization heuristic for constrained task allocation problem. J. Comput. Sci. 7, 37–47.
Rahmani, A.M., Vahedi, M.A., 2008. A novel task scheduling in multiprocessor systems with genetic algorithm by using Elitism stepping method. Sci. Res. Branch.
Raquel, C.R., Naval Jr, P.C., 2005. An effective use of crowding distance in multiobjective particle swarm optimization. Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 257–264.
Sathappan, O., Chitra, P., Venkatesh, P., Prabhu, M., 2011. Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system. Int. J. Inf. Technol. Commun. Converg. 1, 146–158.
Savino, M.M., A., M., 2015. Kanban-driven parts feeding within a semi-automated O-shaped assembly line: a case study in the automotive industry. Assem. Autom. 35, 3–15.
Savino, M.M., Brun, A., Mazza, A., 2014a. Dynamic workforce allocation in a constrained flow shop with multi-agent system. Comput. Ind. 65, 967–975.
Savino, M.M., Mazza, A., Neubert, G., 2014b. Agent-based flow-shop modelling in dynamic environment. Prod. Plan. Control 25, 110–122.
Savino, M.M., Meoli, E., Luo, M., Wong, M.M., 2010. Dynamic batch scheduling in a continuous cycle-constrained production system. Int. J. Serv. Oper. Inform. 5, 313–329.
Singh, J., Singh, G., 2012. Improved task scheduling on parallel system using genetic algorithm. Int. J. Comput. Appl. 39, 17–22.
Topcuoglu, H., Hariri, S., Wu, M.-y., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. Parallel Distrib. Syst. IEEE Trans. on 13, 260–274.
Ullman, J.D., 1975. NP-complete scheduling problems. J. Comput. Syst. Sci. 10, 384–393.
Van Laarhoven, P.J., Aarts, E.H., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. Oper. Res. 40, 113–125.
Wang, G., Wang, Y., Liu, H., Guo, H., 2016. HSIP: a novel task scheduling algorithm for heterogeneous computing. Sci. Program., 2016.
Wang, J., Duan, Q., Jiang, Y., Zhu, X., 2010. A new algorithm for grid independent task schedule: genetic simulated annealing. World Automation Congress (WAC), 2010. IEEE, pp. 165–171.
Xu, Y., Li, K., Hu, J., Li, K., 2014. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf. Sci. 270, 255–287.
Yang, X.-S., Cui, Z., Xiao, R., Gandomi, A.H., Karamanoglu, M., 2013. Swarm Intelligence and Bio-inspired Computation: Theory and Applications. Newnes.
Yang, Y., Wu, G., Chen, J., Dai, W., 2010. Multi-objective optimization based on ant colony optimization in grid over optical burst switching networks. Expert Syst. Appl. 37, 1769–1775.
Zhang, J.J., Hu, W.W., Yang, M.N., 2014. A heuristic greedy algorithm for scheduling out-tree task graphs. TELKOMNIKA Indones. J. Electr. Eng., 12.
Zhang, L., Chen, Y., Sun, R., Jing, S., Yang, B., 2008. A task scheduling algorithm based on PSO for grid computing. Int. J. Comput. Intell. Res. 4, 37–43.
Zhang, L., Li, K., Xu, Y., Mei, J., Zhang, F., Li, K., 2015. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. Inf. Sci. 319, 113–131.
Zomaya, A.Y., Ward, C., Macey, B., 1999. Genetic scheduling for parallel processor systems: comparative studies and performance issues. Parallel Distrib. Syst. IEEE Trans. on 10, 795–812.
Zuo, X., Zhang, G., Tan, W., 2014. Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. Autom. Sci. Eng. IEEE Trans. on 11, 564–573.