The Journal of Systems and Software 124 (2017) 1–21

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing

Bahman Keshanchi[a], Alireza Souri[a], Nima Jafari Navimipour[b,*]

[a] Young Researchers and Elite Club, Tabriz Branch, Islamic Azad University, Tabriz, Iran
[b] Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

ARTICLE INFO

Article history:
Received 26 November 2015
Revised 3 June 2016
Accepted 5 July 2016
Available online 16 July 2016

Keywords:
Cloud computing
Directed acyclic graph
Genetic algorithm
Formal verification
Task scheduling
Model checking

ABSTRACT

Cloud computing is a new platform to manage and provide services on the internet. Lately, researchers have paid attention a lot to this new subject. One of the reasons to have high performance in a cloud environment is the task scheduling. Since the task scheduling is an NP-Complete problem, in many cases, meta-heuristics scheduling algorithms are used. In this paper to optimize the task scheduling solutions, a powerful and improved genetic algorithm is proposed. The proposed algorithm uses the advantages of evolutionary genetic algorithm along with heuristic approaches. For analyzing the correctness of the proposed algorithm, we have presented a behavioral modeling approach based on model checking techniques. Then, the expected specifications of the proposed algorithm is extracted in the form of Linear Temporal Logic (LTL) formulas. To achieve the best performance in verification of the proposed algorithm, we use the Labeled Transition System (LTS) method. Also, the proposed behavioral models are verified using NuSMV and PAT model checkers. Then, the correctness of the proposed algorithm is analyzed according to the verification results in terms of some expected specifications, reachability, fairness, and deadlock-free. The simulation and statistical results revealed that the proposed algorithm outperformed the makespans of the three well-known heuristic algorithms and also the execution time of our recently meta-heuristics algorithm.

1. Introduction

Nowadays, Information Technology (IT) and Internet affect everything from communication to management and industry to businesses (Navimipour and Zareie, 2015; Zareie and Jafari Navimipour, 2016; Zareie and Navimipour, 2016; Navimipour and Soltani, 2016; Soltani and Navimipour, 2016; Charband and Navimipour, 2016; Jafari Navimipour and Charband, 2016). Recently, cloud computing has emerged as a new Internet-based model for enabling convenient, on-demand network access to a shared pool of configurable resources which can be quickly supplied and released with minimal management effort or cloud provider interaction (Milani and Navimipour, 2016; Chiregi and Jafari Navimipour, 2016; Chiregi and Navimipour, 2016). Users want to pay a price which corresponds to the budget they can have (Zeng et al., 2015). Cloud environments contain a variety of resources and services such as storage, processing, memory, network bandwidth and virtual machines. Software as a Service (SaaS) (Alkhanak et al., 2015), Infrastructure as a Service (IaaS) (Malawski et al., 2015), Platform as a Service (PaaS) and Expert as a Service (EaaS) (Jafari Navimipour et al., 2015; Navimipour, 2015) are the most important provided services in the cloud.

Task scheduling on Heterogeneous Distributed Computing Systems (HeDCSs) such as cloud environments is an interesting issue. In the literature, the scheduling problem for both dependent and independent tasks is a well-studied discipline (Tripathy et al., 2015; Ashouraie and Jafari Navimipour, 2015). In the resource management process, before scheduling step, the dependent application should partition by data partitioning into subtasks (Navin et al., 2014). Then, all partitioned subtasks are shown as a DAG, where each node in the DAG shows a subtask and each edge exhibits the communication relationship among the two subtasks (Jun et al., 2013). Finally, subtasks are assigned by the task scheduler to proper resources in distributed systems and task scheduling algorithms decide the execution order of these subtasks (Khan, 2012; Jafari Navimipour et al., 2014). Inasmuch as the assignment of the processors to the subtasks is a challenging issue, in order to minimize makespan of applications, various algorithms have been

* Corresponding author. Fax: +984134203292.
E-mail address: jafari@iaut.ac.ir (N.J. Navimipour).

http://dx.doi.org/10.1016/j.jss.2016.07.006
0164-1212/

proposed (Xu et al., 2014). Since the task scheduling problem on DCS is an NP-complete problem (Navimipour and Khanli, 2008; Kenli et al., 2014), therefore, various studies have been made to obtain near optimal solution for solving this problem (Panda and Jana, 2015).

Genetic algorithm (GA) is a popular type of evolutionary algorithms which is used to optimize complex problems and was introduced in 1975 by Holland (1975). In the GA, an initial random solution is optimized during the generations and each new population is constructed by the crossover and mutation of the individuals in the earlier generation (Mirjalili et al., 2014). With respect to its advantages in the optimization problem, this paper proposed a priority-based task scheduling on heterogeneous environments such as cloud environment based on this algorithm. The proposed algorithm uses the advantages of evolutionary GA algorithm along with heuristic approaches.

On the other hand, there are several techniques for verifying software and hardware systems such as theorem proving (Loveland, 1978), equivalence checking (Huang and Cheng, 1998), type checking and model checking (Clarke et al., 1999). Model Checking is an automated formal verification technique for evaluating functional and non-functional properties of software and hardware systems (Ouchani and Debbabi, 2015; Zhang et al., 2014; Souri et al., 2012). There are many popular model checkers such as SPIN[1], NuSMV[2], UPPAAL[3], and PAT[4]. The classical algorithm for model checking demonstrates and explores the reachable state space of the system using Binary Decision Diagrams (BDDs) (Al-Saqqar et al., 2015). In the few past years, the researchers have been trying to challenge this matter with offering model-driven methods based on formal methods (Rozier, 2011; Jeffery et al., 2015; Al-Saqqar et al., 2016).

In this paper, to optimize task scheduling solutions on HeDCSs such as cloud environments a new genetic algorithm is proposed which is named N-GA. In order to optimize solutions, NGA algorithm first initializes the population. In the next step, assignment of the subtasks to processors is performed and also the fitness value of each chromosome is evaluated in this step to check termination condition and to select chromosome by elitism selection policy. Then, the heuristic single point crossover and mutation operators are applied to the selected chromosomes. Because of formal verification methods have an important role in development processes of the systems, we have present a behavioral modeling approach based on model checking techniques for analyzing the correctness of the proposed algorithm. We extracted the expected specifications of the proposed algorithm in the form of Linear Temporal Logic (LTL) formulas. To achieve the best performance in verification of the proposed algorithm, we use the Labeled Transition System (LTS) method. Also, we verify the proposed behavioral models using NuSMV and PAT model checkers. Then, the correctness of the proposed algorithm is analyzed according to the verification results in terms of some expected specifications, reachability, fairness, and deadlock-free. The simulation and statistical results revealed that the proposed algorithm outperformed the makespans of the three well-known heuristic algorithms and also the execution time of our recently meta-heuristics algorithm.

The four significant contributions of this paper are listed below:

- We used a powerful one-point crossover operator which doesn't allow violating the precedence constraints in produced solutions.
- In order to increasing the convergence of the solutions, elitism technique with unusual selections is adopted to evade premature convergence.
- Statistical analyzes on the different randomly generated graphs are done for ensuring validation of data which obtained through the implementation.
- We translated the proposed technique into a verifiable behavioral model and implemented the behavior model using the PAT and NuSMV model checkers.

The remainder of this paper is organized as follows: in Section 2 related works of different scheduling algorithms on DCSs and some formal verification mechanisms are reviewed. Section 3 presents the proposed N-GA algorithm. Section 4 presents the model checking for the N-GA mechanism. Section 5 describes the experimental and statistical results. Eventually, Section 6 concludes this study. Also, Table 1 shows the used terminologies in this work.

Table 1
The used terminologies in this paper.

Terminology	Description
t_i	The ith subtask in the DAG
p_k	The kth processor in the system
T	Set of subtasks in the DAG
P	Set of processors in a heterogeneous computing system
N	Number of subtask in the DAG
E	Number of edge in the DAG
M	Number of processors in a heterogeneous computing system
t_{entry}	The entry subtask with no predecessor
t_{exit}	The exit subtask with no successor
$Succ(t_i)$	The set of immediate successors of subtask t_i
$Pred(t_i)$	The set of immediate predecessors of subtask t_i
$\overline{W(t_i)}$	The average computational cost of subtask t_i
$W(t_i, p_k)$	The computational cost of subtask t_i on processor p_k
$C(t_i, t_j)$	The amount of communication between subtasks t_i and t_j
$rank_b(t_i)$	The upward rank of subtask t_i
$rank_t(t_i)$	The downward rank of subtask t_i
$Rank_{b+t}(t_i)$	Summation of $rank_b(t_i)$ and $rank_t(t_i)$
$EST(t_i, p_k)$	The earliest start time of subtask t_i on processor p_k
$EFT(t_i, p_k)$	The earliest finish time of subtask t_i on processor p_k
$AST(t_i, p_k)$	The actual start time of subtask t_i on processor p_k
$AFT(t_i, p_k)$	The actual finish time of subtask t_i on the fittest processor p_k
$avail\{k\}$	The earliest time when processor p_k is ready for execution
pop	Abbreviation of population
PopSize	Abbreviation of population size
ChSize	Abbreviation of chromosome size
SLR	Scheduling length ratio of a task graph
CCR	The communication to computation ratio

2. Related work

In this section, firstly, we will introduce the related works in scheduling issues area then we will describe related works of the formal verification methods.

2.1. Scheduling

Scheduling is the assignment of the resources to execute a set of tasks over a period of time (Nasiri, 2015). One of the goals in the task scheduling is the minimizing of the application's makespan (Habibizad Navin et al., 2014). Hence, the scientists intend to find algorithms which should find an optimal assignment of a set of subtasks onto a set of processors while the makespan of application is minimized. Generally, the task scheduling problem has proven to be an NP-complete problem (Ullman, 1975), which refer to the needed time to explore the optimal solutions which enlarge exponentially when the problem size increases (Xu et al., 2013). Therefore, many heuristic and meta-heuristic algorithms have proposed up to now. In recent years, task scheduling is usually done

[1] http://spinroot.com
[2] http://nusmv.fbk.eu/
[3] http://www.uppaal.org/
[4] http://pat.sce.ntu.edu.sg/

by meta-heuristic methods instead of heuristic algorithms which result in a near-optimal solution in polynomial time. Heuristic algorithms search a path in the solution space at the expense of ignoring some possible paths (Xu et al., 2014). The majority of the existing heuristic scheduling algorithms are grouped into list scheduling, duplication, and clustering based scheduling (Khan, 2012). A schedule in the list scheduling is done by two main phases. First of all, priorities are assigned to the subtasks in order to be processed in a sequence. In the next phase, based on the priority, the subtask is assigned to the processors. The priorities are usually assigned based on the computation and communication costs of the application, such as heterogeneous earliest finish time (HEFT) (Topcuoglu et al., 2002) and critical path on a processor (CPOP) (Topcuoglu et al., 2002). On the other hand, the duplication based algorithms try to duplicate the subtasks in order to minimize the makespan. In the heterogeneous systems, the duplication would impressively reduce the makespan of the application as the communication cost is eliminated by placing the subtasks on the same processor (Khan, 2012), such as contention-aware duplication (CA-D) scheduling algorithm (Sinnen et al., 2011). In clustering algorithms, collections of tasks termed as clusters which are mapped to the appropriate processors, such as MLA (Ucar et al., 2006). Usually, clustering algorithms work for homogeneous systems with an unlimited number of processors. The clusters are then scheduled on an available number of processors (Khan, 2012).

Obviously, the list scheduling based algorithms don't consider the global view of the task graph and also, the duplication based algorithms produce a large overhead and are considered inappropriate for systems with low communication cost. Also, the clustering algorithms include the complicated merging phases and as a result, the complexity of the algorithm and its implementation are increased (Khan, 2012). The performance of these algorithms is heavily dependent on the effectiveness of the heuristics. When the complexity of the task scheduling problem enlarges these algorithms don't produce consistent results.

Unlike the heuristic-based algorithms, meta-heuristic based algorithms use random choices to guide themselves through the problem space (Gupta et al., 2010). A genetic algorithm is the popular meta-heuristic algorithm for several kind of the task scheduling problems. Guided random search algorithms such as H2GS (Daoud and Kharma, 2011) combines heuristic algorithm with the meta-heuristic to search for solutions, which has better performance than heuristic algorithms but instead has high complexity against them (Xu et al., 2014; Keshanchi and Jafari Navimipour, 2015).

2.2. Formal verification

Formal methods depict the verification and specification of the systems by using logical and mathematical techniques (Souri and Norouzi, 2015). Particularly, formal techniques use to perform different development activities such as requirement definition and analysis, modeling and model transformation, testing and property verification. So, formal verification of software and hardware systems is an important subject in the many researches.

Yeung (2011) has presented a formal method to address the potential behavioral problems of multi-agent systems for manufacturing control applications. The proposed approach accommodated the need of fine-tuning an application's performance through adjustments in the detailed logic and timing parameters of the agent negotiation protocol. The scalability of the approach is supported by an industrial-strength model checking tool.

Bentahar et al. (2013) have studied symbolic model checking of composite Web services using operational and control behaviors. They addressed the issue of verifying if composite Web services design meets some desirable properties in terms of deadlock freedom, safety, and reachability. They modeled composite Web services based on a separation of concerns between business and control aspects of Web services. This separation was achieved through the design of an operational behavior to define the composition functioning according to the Web services' business logic. Also, it identified the valid sequences of actions that the operational behavior should follow. These two behaviors are formally defined using automata-based techniques. The authors also demonstrated that the proposed method provides checking the soundness and completeness of the model with respect to the operational and control behaviors. Automatic translation procedures from the design models to the NuSMV model checker's code and a verification tool are stated.

Souri and Jafari Navimipour (2014) have proposed an adapted resource discovery approach to address multi-attribute queries in grid computing. They presented a behavioral model for their proposed approach that separate into data gathering, discovery and control behaviors. So, they used the Kripke structure for modeling these behaviors and verified their behavioral models by using NuSMV model checker.

In Safarkhanlou et al. (2015), the authors have proposed a protection service model for an antivirus system. Their model has been focused on preserving in secure state of the antivirus system and translated the proposed model to temporal logic properties by using Computing Tree Logic language. Their proposed model converted to a Kripke Structure. For proving the correctness and the reachability of the proposed model, the authors verified some properties of the proposed model by using NuSMV model checker.

3. Proposed algorithm

In this section, we first present the assumed cloud model and then the N-GA algorithm is described in details.

3.1. Cloud model

The cloud application model inspired from application model in Tang et al. (2015). In the proposed algorithm, the system has a set of P heterogeneous processors which are fully interconnected with a high-speed network. To make task scheduling model simple, it is assumed that the inter-processor communications run at the same speed (i.e., with the same bandwidth) on all links (Xu et al., 2014). The communication cost between two dependent subtasks in application graph must be considered if they aren't assigned to same processors. The task scheduling application is illustrated by DAG where G=(V, E), where V nodes indicate subtasks, and E edges represent dependencies between the subtasks as shown in Fig. 1. The edges of the graph are labeled with communication cost. Each subtask in the application must be run only on one processor and all of the subtasks must be scheduled. In Fig. 1, subtask t_0 is the entry node and subtask t_{10} is the exit node. The unconnected model is considered for the proposed algorithm. So, according to the computation costs, as shown in Table 2 for the DAG in Fig. 1, a processor can be faster for some tasks while being slower for some others. In addition, a node in an application can start when all its predecessors have completed their execution. For instance, subtask t_7 cannot start until t_3 and t_4 complete their execution. First column in Table 2 illustrates the subtasks of the application from Fig. 1, and columns 2 up to 4 present the computation cost of each task on the processors (i.e., p_0, p_1, p_2) and last column (i.e., $\bar{w}$) shows the average computation cost of each tasks on the all processors.

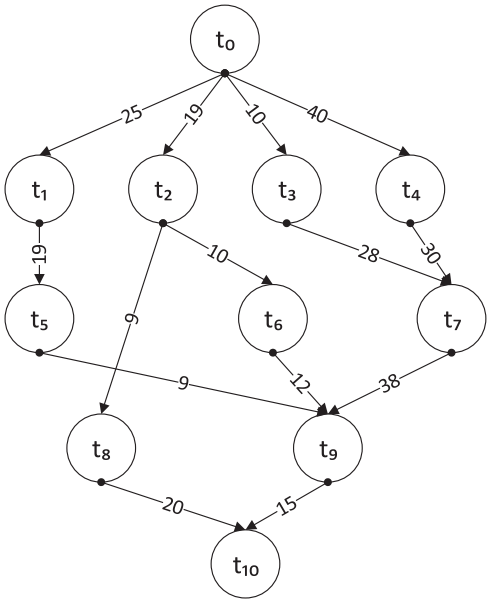


Fig. 1. A simple DAG with 11 nodes and 14 edges.

Table 2
Computation cost matrix of DAG in Fig. 1.

Task	p_0	p_1	p_2	$\bar{w}$
t_0	10	11	12	11
t_1	11	12	13	12
t_2	12	8	13	11
t_3	14	10	18	14
t_4	27	20	19	22
t_5	15	12	18	15
t_6	9	14	19	14
t_7	19	12	14	15
t_8	14	10	15	13
t_9	15	12	15	14
t_{10}	18	10	17	15

3.2. Proposed algorithm

GA is the one of the effective optimization algorithms which is used in complicated problems (Navimipour and Rahmani, 2009). Genetic and evolution mechanisms in the natural systems are the base of the GAs (Abo-Hammour et al., 2011). The GA algorithm as population-based meta-heuristics has various operators which produce the optimized individuals from an initial random population. The GA operators are essential to the convergence of the individuals of a population (Abed et al., 2014). The GA algorithm in the base form has three basic genetic operations (i.e., selection, crossover, and mutation). Some solutions are selected by selection operator from the population as parents and crossover operator crossbreeds the parents to produce offspring and mutation operator changes the offspring in accord with the mutation rules. Solutions are called individuals or chromosomes in the GA, and iterations of the GAs are called generations (Xu et al., 2014). Fig. 2 presents the steps on the proposed GA algorithm by flowchart and the main function of this algorithm is shown in Algorithm 1.

3.3.1. Initial population

In order to create chromosomes, encoding is the first step. In the proposed algorithm, a chromosome constructs by a permutation of integer numbers 0, 1, 2... n-1 that illustrates priority order of the n gene (subtasks) in an application, as shown in Fig. 3. Each chromosome in population implies a solution of the problem. Besides, the subtasks in the chromosome must have a valid topological order, in a way that the entry node should be placed at the first location of the chromosome, while the exit node should be placed at the last location of the chromosome and all the subtasks in an application must be scheduled and the schedule must not violate the precedence constraints. PopSize implies to the size of the initial population and is equal to $4 \times n$ where never changes through the generation. In order to has a good "seeding" in the initial population of proposed algorithm, three heuristic rank policies(i.e. upward rank Eq. (1), downward rank Eq. (2) and a combination of

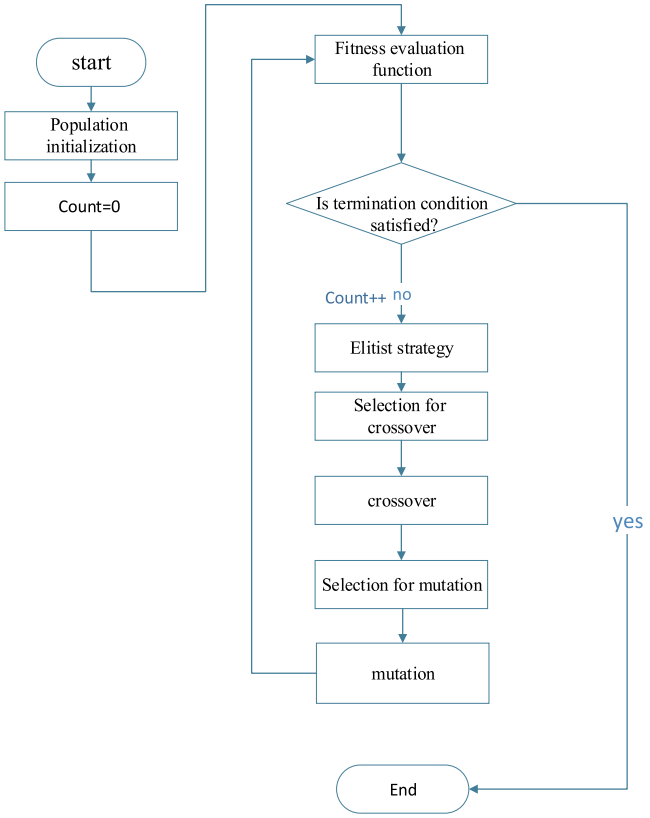


Fig. 2. The N-GA flowchart.

Algorithm 1. Main algorithm.

Input:
Parameters for the genetic algorithm;
Parameters for task scheduling.
Output:
A task schedule.
1: Call **Algorithm 2** to initialize a population;
2: Call **Algorithm 3** to assign subtasks to processors and evaluate the fitness of chromosomes;
3: **while** the termination condition is satisfied
4: Import elitism chromosomes to the new population
5: Call **Algorithm 4** to perform selection and the single point crossover operator;
6: Call **Algorithm 5** to perform selection and mutation operator;
7: **end while**
8: **return** an optimized schedule.

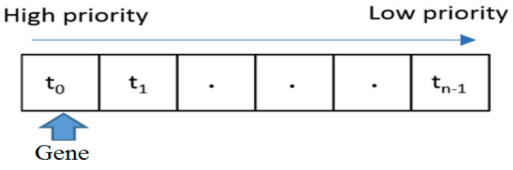


Fig. 3. Encoding structure of chromosome in the N-GA.

Table 3
Subtask ranks of the DAG in the Fig. 1.

Subtask (t_i)	$Rank_b$	$Rank_t$	$Rank_{b+t}$	Level
t_0	200	0	200	0
t_1	99	36	135	1
t_2	91	30	121	1
t_3	139	21	160	1
t_4	149	51	200	1
t_5	68	67	135	2
t_6	70	51	121	2
t_7	97	103	200	2
t_8	48	50	98	2
t_9	44	156	200	3
t_{10}	15	185	200	4

Algorithm 2. Initial population.

Input:
PopSize;
ChSize;
Output:
Random generated population
1: Initial three of the chromosomes by using three heuristic rank policies;
2: **for** i=3 to PopSize-1 do
3: **for** j=0 to ChSize-1 do
4: generate a gene j $\in (0, ChSize-1)$randomly that not generated in pervious genes;
5: **end for**
6: move gene i from left to right in first valid place in the queue in order to have a valid topological order;
7: **end for**

level Eq. (3) and upward-downward rank) are used, which recently they had been used frequently for giving the priority to each subtask in DAG (Xu et al., 2014). Table 4 shows the subtasks priority queues of three mentioned heuristic with using the ranks values in Table 3. These three heuristic have initialized three chromosomes of the population in the proposed algorithm. The rest of the chromosomes in population are initialized by random perturbation which they are sorted from left to right for preserving the precedence constraints of subtasks in an application. Algorithm 2 shows the initial population process in the proposed algorithm.

$$rank_b(t_i) = \overline{W(t_i)} + \max_{t_j \in succ(t_i)} \left(C(t_i, t_j) + rank_b(t_j)\right) \quad (1)$$

Where the $\overline{W(t_i)}$ is the average computational cost of subtask t_i, $C(t_i, t_j)$ is the quantity of the communication between the subtasks t_i and t_j and $rank_b(t_j)$ is the upward rank of the subtask t_i's successor.

$$rank_t(t_i) = \max_{t_j \in pred(t_i)} \left(rank_t(t_j) + \overline{W(t_j)} + C(t_i, t_j)\right) \quad (2)$$

Where $rank_t(t_j)$ is the downward rank of the subtask t_i's precedence.

$$Level(t_i) = \begin{cases} 0, & \text{if } t_i = t_{entry}; \\ \max(Level(t_j)) + 1, & \\ \quad t_j \in pred(t_i) & \text{otherwise} \end{cases} \quad (3)$$

Where t_j is the subtask t_i's precedence.

3.3.2. Assigning subtasks to processors

Usually in the initial population, each chromosome must be a valid priority permutation which means that the subtasks must not violate precedence constraints in an application. In each chromosome, a gene is ready to be assigned to the processors which have the highest priority and also it is unscheduled. In the proposed scheduling algorithm, the HEFT method (Topcuoglu et al., 2002) is employed. The HEFT method picks out subtask which has the highest priority in the chromosome and then assigns it to the processor

Algorithm 3. Assigning subtasks to processors function.

Input:
The current population chromosomes.
Output:
Makespan
1: Fill the priority queue with subtasks;
2: **while** the priority queue is not empty **do**
3: Select the first subtask t_i from the priority queue;
4: **for** each processor p_k in the processor set **do**
5: Compute $EFT(t_i, p_k)$ value using the insertion-based HEFT scheduling policy;
6: Assign subtask t_i to the processor p_k that minimizes $EFT(t_i, p_k)$;
7: **end for**;
8: Remove t_i from the priority queue;
9: **end while**;
10: **return** makespan=$AFT(t_{exit})$.

to minimize its computation time while using the insertion-based scheduling policy.

The earliest start time (EST) of the subtask t_i on processor p_k is symbolized as EST (t_i, p_k) which is obtained by Eqs. (4) and (5).

$$EST(t_{entry}, p_k) = 0 \quad (4)$$

$$EST(t_i, p_k) = \max_{t_j \in pred(t_i)} \left(AFT(t_j) + C(t_j, t_i)\right) \quad (5)$$

The actual start time of the subtask t_i on processor p_k is symbolized as AST (t_i, p_k) which is obtained by Eq. (6).

$$AST(t_i, p_k) = max(EST(t_i, p_k), Avail\ (p_k)) \quad (6)$$

Where the $Avail\ (p_k)$ is a time that the processor p_k has idle and ready for the task execution.

The earliest finish time of the subtask t_i on processor p_k is symbolized as $EFT(t_i, p_k)$ which is obtained by Eq. (7).

$$EFT(t_i, p_k) = W(t_i, p_k) + AST(t_i, p_k) \quad (7)$$

Where $W(t_i, p_k)$ is the computational cost of the subtask t_i on processor p_k.

The actual finish time of the subtask t_i on processor p_k is symbolized as $AFT(t_i, p_k)$, where the p_k is the proper processor for the subtask t_i which is obtained by Eq. (8).

$$AFT(t_i, p_k) = \min_{1 \le l \le P} EFT(t_i, p_k) \quad (8)$$

Assigning subtasks to processors is described in Algorithm 3.

3.3.3. Fitness function

Fitness value determines which chromosomes would be survived to generate the next-generation population. Like to the all scheduling algorithm, the makespan of the application in the proposed algorithm is equal to the latest finish time among all subtasks, which is equivalent to the actual finish time of the exit subtask. The makespan is obtained using Eq. (9).

$$makespan = AFT(t_{exit}) \quad (9)$$

In the N-GA, the fitness value of chromosome is acquired by Eq. (10).

$$fitness_i = makspan_i \quad (10)$$

The low fitness value of the $chromosome_i$ represents that it has a short makespan. This means that the highest and the lowest fitness in the population are worst and best chromosomes in the population respectively.

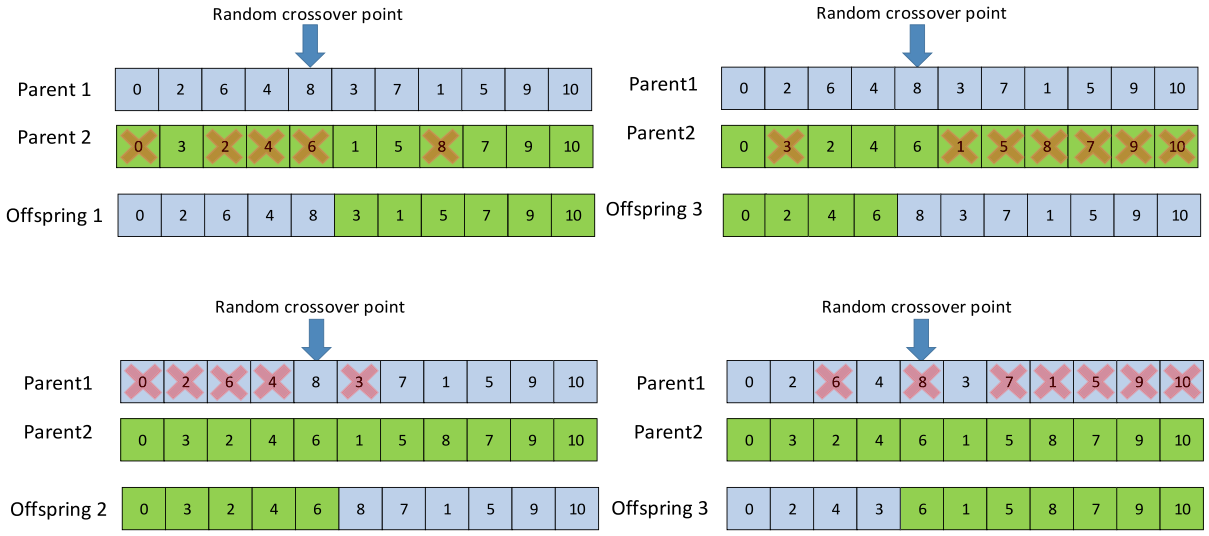


Fig. 4. Single-point crossover operation.

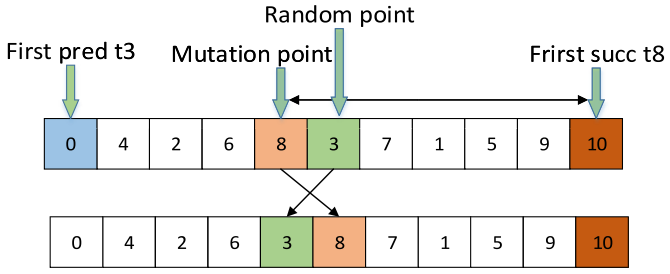


Fig. 5. Mutation operator.

3.3.4. Crossover

Crossover operator has a significant role in the GA algorithms which is used to provide variation in chromosomes of the population. Besides, crossover operator helps to evolve the population. In our recent research (Keshanchi and Jafari Navimipour, 2015), we proposed a new heuristic crossover operator which it mostly changes the solutions in the population and could discover 4 diverse offspring from two parents which the produced offspring create diversity on both sides of the single crossover point in parents as shown in Fig. 4. We had replaced the offspring with parents if they optimized the fitness of the parents which this comparison causes more complexity in our recent algorithm. Therefore, in this work we have not use comparison and we replace the offspring with parents according to the Algorithm 4 where presents N-GA crossover operator in more detail.

3.3.5. Mutation operator

In the GA, mutation operator replaces a gene with another gene in the same chromosome with a certain probability. Mutation operator helps to impose variety and diversity in the population. Additionally, this operator broadens the search space and lets the algorithm escape from local optimum. In this work, the recently proposed powerful mutation is used (Xu et al., 2014). This operator avoids the algorithm to violate precedence constraints. According to the Fig. 5, in the first step, a gene t_i (in the example is the t_8) is selected randomly from a chromosome. Then, the first successor for the task t_i from the mutation point to the end of the chromosome is acquired which is the t_{10}. If there is any gene in $[i+1, j-1]$ and the precedencies of t_j are not in front of t_i, t_i and t_j can be replaced with each other which is shown in the Fig. 5. If these conditions do not occur, so, the mutation operator will be run from the first step. Algorithm 5 shows the mutation operator in details.

3.3.6. Termination condition

The fundamental difference between natural evolution and artificial evolution in complex problem solving is the natural evolution which species do not usually tend for termination. On the other hand, in the problem solving, we purposely need to stop the generation of the process (Ong and Fukushima, 2011). There are some approaches such as setting a limitation on the number of fitness evaluations or the computer clock time, or to trace the population's diversity and stop when this falls below a preset threshold for terminating evolutionary algorithms (Xu et al., 2014). In the N-GA, the termination condition occurs when all the chromosomes are converged into the same fitness.

3.3.7. Example

In this section, the simple DAG in Fig. 1 is scheduled by three heuristic, memetic algorithm and the proposed algorithms. Fig. 6 presents the five solutions of scheduling the DAG in Fig. 1 by the HEFT-B (Topcuoglu et al., 2002), the HEFT-T (Topcuoglu et al., 2002), the HEFT-L (Xu et al., 2014), the MPQMA and N-GA task scheduling algorithms respectively. The task scheduling priority queues in Table 4 are used by HEFT-B, HEFT-T and HEFT-L algorithms and the priority queues for the MPQMA (t_0,t_4,t_1,t_3,t_2,t_8,t_6,t_7,t_5,t_9,t_{10}) and N-GA (t_0,t_4,t_2,t_8,t_6,t_3,t_1,t_7,t_5,t_9,t_{10}) are chosen randomly among the final forty solutions. Fig. 7 shows the makespans of these five algorithms in the DAG on the Fig. 1. Results of the solutions reveal that the N-GA algorithm outperformed the makespans of the three heuristic algorithms while it produced the same makespan like MPQMA.

4. Model checking for the N-GA mechanism

Formal verification is used to a mathematical demonstration of the correctness of a system behavior. Formal verification methods such as model checking and theorem proving have an important role in validation processes of the systems. The model checking technique is marvelous because of its simplicity with a compact theoretical foundation on the verification approach. While it does not require high skilled specialists as usually needed for theorem proving, it offers a very expressive resource for properties specification through temporal logic. Also, model checking is an impor-

Algorithm 4. Crossover operator.

Input:
The current population chromosomes.
Output:
New population
1: bool flag=true;
2: **for** j=0 to (PopSize × 80/100) **do** // 80 percent of population
3: **if**((PopSize × 80/100)*20/100) **then**
4: par1= Generate random number between 0 to PopSize-1 except elite chromosomes
5: par2= Generate random number between 0 to PopSize-1
6: **end if**
7: **else**
8: par1=Generate random number between 0 to PopSize-1 except elite chromosomes
9: par2=Generate random number between 0 to PopSize-1 except elite chromosomes
10: **end else**
11: Generate randomly a suitable crossover point i;
12: Cut the parent1's and the parent2's chromosomes into left and right segments;
13: Generate a new offspring, namely the offspring1;
14: Inherit the left segment of the parent1's chromosome to the left segment of the offspring1;
15: Copy genes in parent2's chromosome that does not appear in the left segment of parent1's chromosome to the right segment of offsring1's chromosome;
16: Generate a new offspring, namely the offspring2;
17: Inherit the left segment of the parent2's chromosome to the left segment of the offspring2's chromosome;
18: Copy genes in parent1's chromosome that does not appear in the left segment of parent2's chromosome to the right segment of offspring2's chromosome;
19: Generate a new offspring, namely the offspring3;
20: Inherit the right segment of the parent1's chromosome to the right segment of the offspring3;
21: Copy genes in parent2's chromosome that does not appear in the right segment of parent1's chromosome to the right segment of offsring3's chromosome;
22: Generate a new offspring, namely the offspring4;
23: Inherit the right segment of the parent2's chromosome to the right segment of the offspring4's chromosome;
24: Copy genes in parent1's chromosome that does not appear in the right segment of parent2's chromosome to the right segment of offspring2's chromosome;
25: **if** (flag == true)**then**
26: **if** (random generate 0 or 1 == 0)**then**
27: **replace** par1 with offspring 1
28: **end if**
29: **else**
30: **replace** par1 with offspring 3
31: **end else**
32: flag=false**;**
33: **end if**
34: **else**
35: **if** (random generate 0 or 1 == 0)**then**
36: **replace** par1 with offspring 2
37: **end if**
38: **else**
39: **replace** par1 with offspring 4
40: **end else**
41: flag=true;
42: **end else**
43: **end for**
44: **return** new population.

Algorithm 5. Mutation operator.

Input:
The current population chromosomes.
Output:
New population
1: A randomly chosen chromosome Ch[i];
2: Choose randomly a gene T_i;
3: Find the first successor $T_j \in Succ(i)$;
4: Choose randomly a gene T_k in the interval $[i+1, j-1]$;
5: **if** l < i for all $T_l \in Pred(k)$ then
6: Generate a new offspring by interchanging gene T_i and gene T_k;
7: return the new offspring;
8: **else**
9: Go to Step 1;

Table 4
Task priority queues.

	Position of gene										
	0	1	2	3	4	5	6	7	8	9	10
HEFT-B priority queue	t_0	t_4	t_3	t_1	t_7	t_2	t_6	t_5	t_8	t_9	t_{10}
HEFT-T priority queue	t_i	t_3	t_2	t_1	t_8	t_4	t_6	t_5	t_7	t_9	t_{10}
HEFT-L priority queue	t_0	t_4	t_3	t_1	t_2	t_7	t_5	t_6	t_8	t_9	t_{10}

tant technique for the verifying the distributed and software systems (Clarke et al., 1999).

This section illustrates a formal method approach for analyzing and verifying N-GA mechanism. This approach includes two methods for implementing the verification methodologies. The deficiency of attention to the formal verification and specification phases reduces the comprehensive sympathetic of the construction and configuration of the proposed system. There are several techniques for verifying software and hardware systems such as theorem proving (Loveland, 1978), equivalence checking (Huang and Cheng, 1998), type checking and model checking (Clarke et al., 1999). Model Checking is an automated formal verification technique for evaluating functional and non-functional properties of software and hardware systems (Ouchani and Debbabi, 2015; Zhang et al., 2014; Souri et al., 2012). There are many popular model checkers such as SPIN[5], NuSMV[6], UPPAAL[7], and PAT[8]. The classical algorithm for model checking demonstrates and explores

[5] http://spinroot.com
[6] http://nusmv.fbk.eu/
[7] http://www.uppaal.org/
[8] http://pat.sce.ntu.edu.sg/

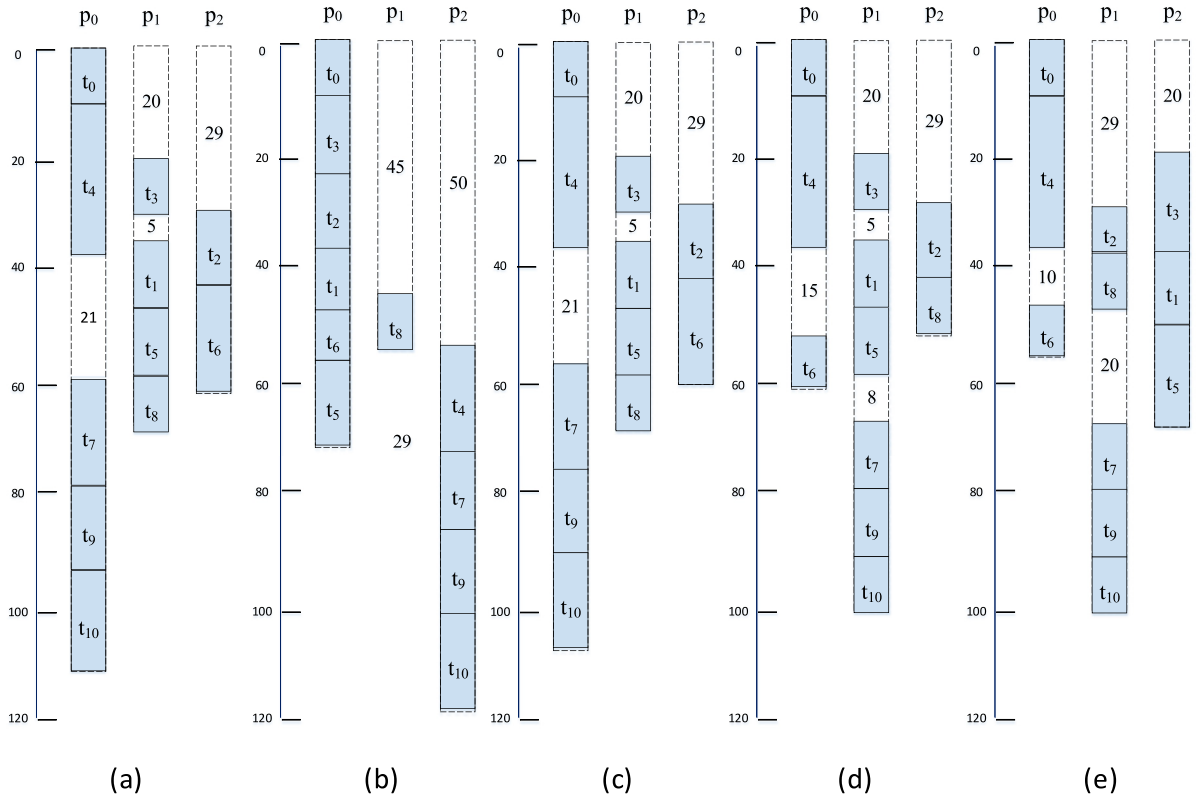


Fig. 6. (a) HEFT-B, (b) HEFT-T, (c) HEFT-L, (d) MPQMA and (e) N-GA scheduling sequence on three processors (Table 2) for the DAG in Fig. 1.

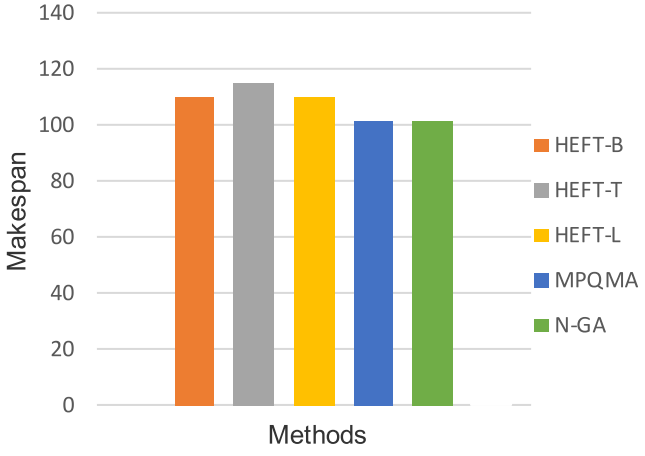


Fig. 7. Makespan of five different algorithms for the DAG in Fig. 1.

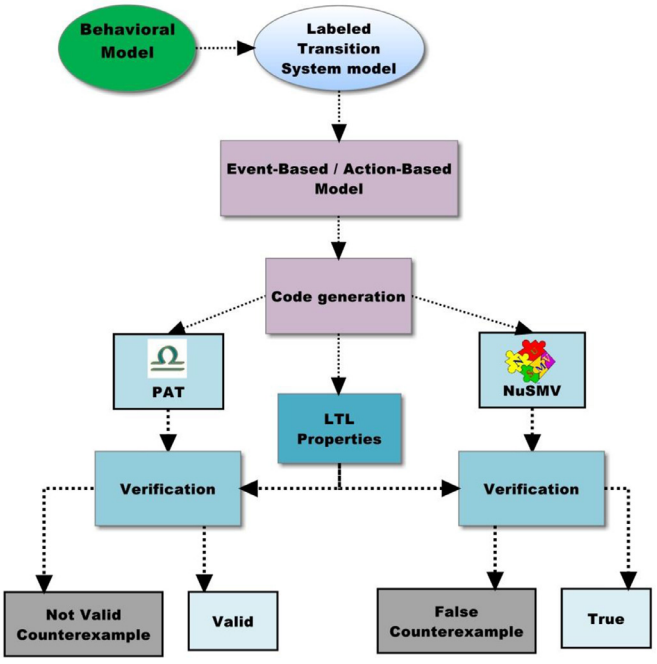


Fig. 8. Model checking architecture for the N-GA mechanism.

the reachable state spaces of the system using Binary Decision Diagrams (BDDs).

In the few past years, the researchers have been trying to challenge this matter with offering model-driven methods based on formal methods. In this section, we use Labeled Transition System (LTS) method for N-GA mechanism. Then, we define the expected properties of the N-GA mechanism through Linear Temporal Logic (LTL) language (Rozier, 2011; Bae and Meseguer, 2015). After specifying the expected properties for verifying the method, the created models will be implemented by NuSMV model checker and Process Analysis Toolkit (PAT) as two powerful tools (Rezaee et al., 2014) to achieve the best performance in verification results. Fig. 8 shows the model checking mechanism for verifying the behavior model of the N-GA mechanism. Firstly, behavioral modeling N-GA mechanism is described in Section 4.1 . Then, Section 4.2 describes the verification method and Section 4.3 illustrates evaluating verification results.

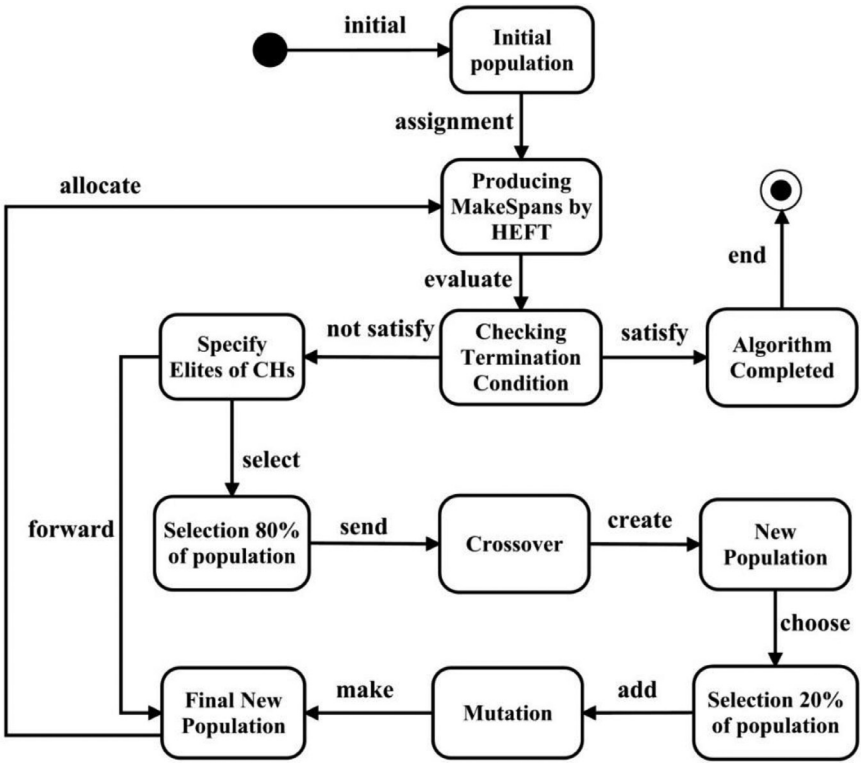


Fig. 9. The behavioral model of the N-GA mechanism.

4.1. Behavioral modeling

In this subsection, we present a behavior model for the proposed mechanism according to Fig. 9. First of all, in the proposed algorithm the parameters for the GA and task scheduling are initialized. Then, the initial population is generated. In the next step, the makespans of the chromosomes by HEFT task assignment approach are obtained. After that, the termination condition by evaluating the finesses of the chromosomes is checked. Then, if the condition is satisfied, the algorithm is terminated otherwise the elitist chromosomes are imported to the new population directly. In the next step, 80% of the population is selected randomly and the selected chromosomes are crossbred. Then, 20% of the chromosomes from the obtained new population are mated by mutation operator. Finally, the final new population is created. This process is looped until the termination condition is being true.

After describing the behavioral model of the N-GA mechanism, we present two verification mapping methods in terms of event-based and action based labeled transition system for our proposed mechanism. A mapping method illustrates the interactions between the actions and the reactions of an LTS behavioral model.

Definition 1. An Event Based Labeled Transition System $\mathscr{ELT}$ is a 4-tuple $\mathscr{ELT}=(\mathscr{S}, s, \mathscr{E}, \mathscr{T})$ where:

- $\mathscr{S}$ is a set of states.
- s is the set of initial state: $s \in \mathscr{S}$.
- $\mathscr{E}$ is a set of events.
- $\mathscr{T}$ is a total transition relation: $\mathscr{T} \subseteq \mathscr{S} \times \mathscr{E} \times \mathscr{S}$. This means, the relation $s_1 \xrightarrow{e} s_2$ ($s_1, s_2 \in \mathscr{S}$ and $e \in \mathscr{E}$) is used for stating that $(s_1, e, s_2) \in \mathscr{T}$.

Definition 2. An Action-Based Labeled Transition System $\mathscr{ALT}$ is a 4-tuple $\mathscr{ALT}=(\mathscr{Q}, q, \mathscr{A}, \mathscr{R})$ where:

- $\mathscr{Q}$ is a set of events.
- q is the set of the initial event: $q \in \mathscr{Q}$.
- $\mathscr{A}$ is a set of actions.
- $\mathscr{R}$ is a total transition relation: $\mathscr{R} \subseteq \mathscr{Q} \times \mathscr{A} \times \mathscr{Q}$. This means, the relation $q_1 \xrightarrow{a} q_2$ ($q_1, q_2 \in \mathscr{Q}$ and $a \in \mathscr{A}$) is used for stating that $(q_1, a, q_2) \in \mathscr{R}$

In the first method, the model is considered as event based. According to N-GA behavioral model, we have:

State=(Start, Init_Pop, Prod_MS_HEFT, Check_Term_Cond, Algorithm_Comp, Specify_Elite_CH, Selection_80, Crossover, New_Pop, Selection_20, Mutation, Final_New_Pop, End);

Event= (initial, assignment, evaluate, satisfied, notsatisfied, forward, select, send, create, choose, add, make, allocate, end);

We can define a path on the $\mathscr{ELT}$ behavioral model of the N-GA mechanism as follow:

Definition 3. An Event Labeled path ***ELP*** is a finite sequence of the states and events starting from state s_1 and finishing at state s_2 (s_1 and $s_2 \in \mathscr{S}$) denoted as:

$\boldsymbol{ELP}=s_1 \xrightarrow{e1} s_2 \xrightarrow{e2} s_3 \ldots s_{n-1} \xrightarrow{en} e_n$ such that $\forall(k, v) : (s_k, e_v, s_{k+1}) \in \mathscr{T}$. For example, in Fig. 9, ***LP(1)=*** *Start* $\xrightarrow{initial}$ *Initial Population* $\xrightarrow{assignment}$ *Producing Make-Span by HEFT* $\xrightarrow{evaluate}$ *Checking Termination Condition* is a path in the N-GA LTS model.

In the second method, the model is considered as action based. According to N-GA behavioral model, we have:

Event=(initial, assignment, evaluate, satisfied, notsatisfied, forward, select, send, create, choose, add, make, allocate, end);

Action= (Init_Pop, Prod_MS_HEFT, Check_Term_Cond, Algorithm_Comp, Specify_Elite_CH, Selection_80, Crossover, New_Pop, Selection_20, Mutation, Final_New_Pop, End);

We can define a path on the $\mathscr{ALT}$ behavioral model of the N-GA mechanism as follow:

Definition 4. An Action Labeled path ***ALP*** is a finite sequence of the states and actions starting from state q_1 and finishing at state q_2 (q_1 and $q_2 \in Q$) denoted as:

ALP $= q_1 \xrightarrow{a1} q_2 \xrightarrow{a2} q_3 \ldots q_{n-1} \xrightarrow{an} q_n$ such that $\forall(k, v) : (q_k, a_v, q_{k+1}) \in \mathscr{R}$

ALP $=$ *evaluate* $\xrightarrow{\text{Checking Termination Condition}}$ *satisfy* $\xrightarrow{\text{Algorithm Completed}}$ *end* is a path in the N-GA LTS model.

According to above definitions, we can prove the correctness of the $\mathscr{ELT}$ behavioral model and the $\mathscr{ALT}$ behavioral model using verification of some expected properties using LTL formulas. By using the Event Labeled Path ***ELP*** and the Action Labeled Path ***ALP***, we create the state space of the proposed N-GA behavioral models in model checking implementation.

After describing the two verification methods, we define some expected properties for verifying our model using LTL language. Linear Temporal Logic is a formal specification language for verifying and describing the expected properties of the systems that are generally used in model checking tools. The formulas of the propositional linear temporal logic, LTL, defined as follow (Clavel et al., 2007; Baier and Katoen, 2008; Jafari Navimipour et al., 2015):

- ***Expected Properties*** (***EP***): if $\alpha, \beta \in$ ***EP*** then $\alpha, \beta \in$ LTL (***EP***).
- ***True***: $\mathscr{T} \in$ LTL (***EP***).
- ***Next*** operator: If $\alpha \in$ LTL (***EP***), then $X\alpha \in$ LTL(***EP***).
- ***General*** operator: If $\alpha \in$ LTL (***EP***), then $G\alpha \in$ LTL(***EP***).
- ***Future*** operator: If $\alpha \in$ LTL (***EP***), then $F\alpha \in$ LTL(***EP***).
- ***Until*** operator: If $\alpha, \beta \in$ LTL (***EP***), then α U $\beta \in$ LTL(***EP***).
- ***Boolean connectives***: If $\alpha, \beta \in$ LTL (***EP***), then the formulas (! α) and (α | β) and (α & β) $\in$LTL (***EP***).

By using these initiations, some expected properties of N-GA model are specified in LTL language. First, we Let $\rightarrow$ as the logical association. There are two LTL properties types in this step because we have two verification methods for N-GA mechanism according to proposed behavior model in Fig. 9.

The following LTL specifications are used to event based model of N-GA mechanism:

- L1 G (N-GA.state = Start & N-GA.event = initial) -> X (N-GA.state = Prod_MS_HEFT & N-GA.event=assignment);
- L2 G (N-GA.event = notsatisfied) -> X (N-GA.event = select | N-GA.event = forward);
- L3 F ((N-GA.state = Mutation) -> X (N-GA.state = Final_New_Pop & N-GA.event=make) U (N-GA.event = evaluate & N-GA.event =satisfied));
- L4 G ((N-GA.state = Selection_80 & N-GA.event = send) -> (N-GA.state = Crossover & N-GA.event=create) -> (N-GA.state = Selection_20 & N-GA.event=choose)-> F(N-GA.state = Mutation));
- L5 G (N-GA.state = Mutation & N-GA.event = make) -> X (N-GA.state = Final_New_Pop & N-GA.event = allocate);
- L6 G (N-GA.event = satisfied) -> X (N-GA.state = Algorithm_Comp & N-GA.event = end);
- L7 G ((N-GA.state = Start & N-GA.event=assignment)-> (N-GA.state = Prod_MS_HEFT & N-GA.event=evaluate)) U (F(N-GA.state = Check_Term_Cond & N-GA.event=satisfied)-> X(N-GA.state = Algorithm_Comp & N-GA.state = End));
- L8 G (N-GA.state = Final_New_Pop & N-GA.event= allocate) -> (N-GA.event= evaluate & N-GA.state = Check_Term_Cond) U (N-GA.event= satisfied & N-GA.event= end);
- L9 G (!(N-GA.state = Mutation) U ((N-GA.state = Selection_80 & N-GA.event = send) -> (N-GA.state = Crossover & N-GA.event=create) -> (N-GA.state = Selection_20 & N-GA.event=choose)));
- The following LTL specifications are used to action based model of N-GA mechanism:
- L1 G (N-GA.state = initial & N-GA.action=Init_Pop)-> X(N-GA.action= Prod_MS_HEFT);
- L2 G ((N-GA.action = Prod_MS_HEFT & N-GA.action=Check_Term_Cond & N-GA.state = notsatisfied) -> F(N-GA.action = Selection_80));

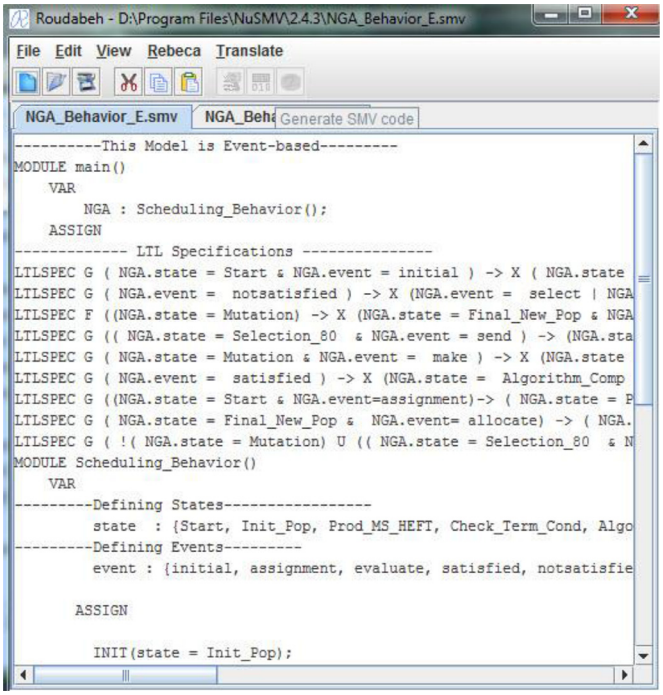


Fig. 10. The snapshot of SMV codes for the N-GA mechanism (event based model).

- L3 G (N-GA.action= Check_Term_Cond & N-GA.state = satisfied) -> X (N-GA.action= Algorithm_Comp & N-GA.state = end);
- L4 G ((N-GA.state = select & N-GA.action= Selection_80 & N-GA.action = Selection_20) -> X (N-GA.action = Mutation & N-GA.state = make));
- L5 G ((N-GA.action = Crossover & N-GA.action= Mutation) -> (N-GA.action = Final_New_Pop & N-GA.state =allocate) U (N-GA.state = satisfied & N-GA.action = Algorithm_Comp));
- L6 G (N-GA.action = Specify_Elite_CH) -> X (N-GA.state = select | N-GA.state = forward);
- L7 G ((N-GA.action = Check_Term_Cond) -> X (N-GA.action = Algorithm_Comp & N-GA.action = Specify_Elite_CH));
- L8 G ((N-GA.action = Prod_MS_HEFT) -> X (N-GA.action= Specify_Elite_CH & N-GA.state = forward));

4.2. Verification methods

To achieve the best performance in verification results, we use the NuSMV model checker and Process Analysis Toolkit (PAT) as implementation platforms for proposed mechanism. First, we convert the proposed Labeled models into *SMV* code. The NuSMV model checker is a formal foundation which is appropriate for modeling concurrent and distributed systems (Al-Saqqar et al., 2015). This model checker has designed in an effort to bridge the gap between formal verification approaches and real applications (El Kholy et al., 2014). It also supports Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) as input languages and NuSMV as an output code (Bentahar et al., 2013, Zhao and Rozier, 2014). Figs. 10 and 11 illustrate the main components of *SMV* coding in two labeled models as event-based and action-based methods respectively. In Fig. 10, we can see the event based model as *SMV* codes. The main module defines *Scheduling_Behavior()* process that includes states definition and events definition. Fig. 11 shows action based model as *SMV* codes. The main module defines *Scheduling_Behavior()* process that includes states and actions definition. Now, we describe the model checking approach for the NuSMV model checker briefly. Firstly, for verifying the proposed model, the *read_model* command is used

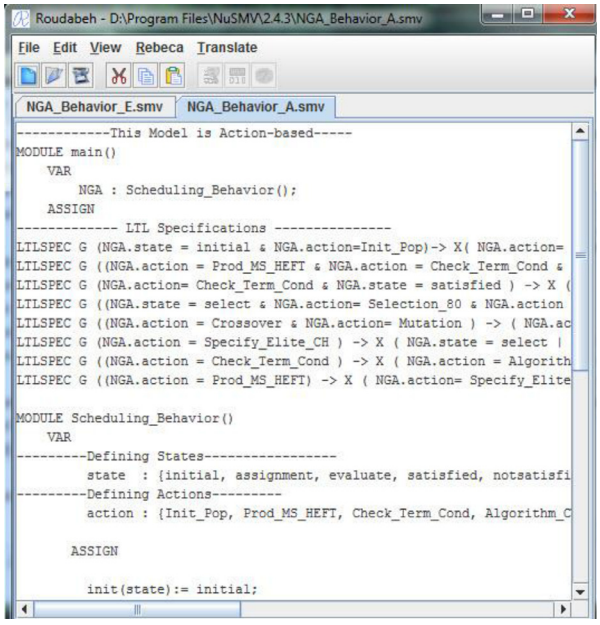


Fig. 11. The snapshot of SMV codes for the N-GA mechanism (action based model).

which receipts a file name as an argument. After the correct execution of this command, an internal demonstration of the delivery file is made and stored. The second step includes the hierarchical description into a flattened description by command *flatten_hierarchy*. The third step consists in the encoding of the scalar variables into Boolean variables using *encode_variables* command. The fourth step consists of the collecting the structures built by the *flatten_hierarchy* into BDDs using the encoding performed by *build_model*. The fifth step is the checking the LTL specifications. These commands can be executed only when all the previous commands have been truly performed.

Also, we implement the proposed labeled models in PAT model checker. Fig. 12 displays the N-GA event-based model in PAT model checker. The first state is *Start* and final state is *End*. Fig. 13 shows the N-GA action based model in PAT model checker. The first event is an *initial* and final event is the *end*.

4.3. Verification results

This subsection illustrates the evaluation of verification results according to implementing two verification methods. We executed NuSMV and PAT model checkers on the PC with an Intel Core i5, 3.33 GHz CPU and 4GB RAM. Fig. 14 displays a statistic schema of model checking time for each the specification in the event-based model method using NuSMV and PAT model checkers. According to the Fig. 14, the model checking the time of NuSMV model checker is lower than the model checking the time of PAT model checker in the event-based model.

Fig. 15 shows the model checking time for each of the specification in action based model using NuSMV and PAT model checkers. According to Fig. 15, the model checking the time of PAT model checker is lower than the model checking the time of the NuSMV model checker in action based model.

Fig. 16 illustrates a snapshot of the NuSMV interactive mode for checking proposed LTL specifications in the event-based model of N-GA mechanism. As shown in the Fig. 16, L1, L2, L3, L4, L5, L6, L7, L8 and L9 as defined in Section 4.2 are true (by the green line in Fig. 16). In addition, Fig. 17 shows a snapshot of the PAT model checker for checking proposed LTL specifications in the event-based model of N-GA mechanism. As shown in the Fig. 17, L1, L2, L3, L4, L5, L6, L7, L8 and L9 as defined in Section 4.2 are true.

Table 5
The values of parameters.

Parameters	Description
The population size	Four times the number of subtasks
The crossover probability	0.8
The mutation probability	0.2
The termination condition	When all the chromosomes converge to the same makespan

In Fig. 18, the verification results of action based model in NuSMV model checker are presented. The L1, L2, L3, L4, L5, and L6 are a true condition (by the green line). However, the L7 and L8 specifications as false condition are not satisfied (by the red line). In addition, Fig. 19 shows verification results of action based model in PAT model checker by same results. The L7 and L8 specifications are false for verification results of the action based model in PAT model checker.

Finally, the verification results show that all of the corrected specifications have been satisfied using NuSMV and PAT model checkers. Also, the uncorrected specifications have been detected by some counterexamples in two model checker. So, all of the expected specification according to the proposed method behavior are satisfied by using our verification methods.

Figs. 20 and 21 describe the correctness of event based and action based behavioral models based on Binary Decision Diagram Method (BDDM) in NuSMV model checker respectively. By using the *compute_reachable* command, we can check state reachability, state fairness, and transition fairness. The command of "print_reachable_states" is performed to identify system diameter (minimum number of iterations of the NuSMV model to obtain all the reachable states) and count the number of reachable states. By using *check_fsm* command, we can see the deadlock-free status in the N-GA model. In Fig. 20, the 92.307% of the event based states are reachable and fair. Also, there is no any deadlock in state space of the event based model. Fig. 20 shows that the system diameter is 10 and there are totally 168 reachable states. Also, the numbers of fair states and fair transitions are 168 that equal by reachable states. In Fig. 21, the 93.452% of the action based states are reachable and fair. Also, there is no any deadlock in state space of the action based model. The Fig. 20 displays that the system diameter is 10 and there are totally 157 reachable states. Also, the numbers of fair states and fair transitions are 157 that equal by reachable states. The correctness results showed that the proposed algorithm is reachable, fair and deadlock-free in two verification methods.

5. Experimental results

To evaluate the N-GA approach, we compared it with three heuristics algorithms, i.e., HEFT-B, HEFT-T (Topcuoglu et al., 2002), HEFT-L that uses level priority of subtasks (Xu et al., 2014) and our recently memetic algorithm which named MPQMA (Keshanchi and Jafari Navimipour, 2015). We have considered the illustrated DAG in Fig. 1 and randomly generated application DAGs to evaluate the proposed algorithms against the other four scheduling algorithm. To evaluate the performance of the N-GA several metrics are used. N-GA is implemented in C# language on Azure cloud environment. The simulations are done on the PC with an Intel Core 5, 3.33 GHz CPU and 4GB RAM. Table 5 represents the values of the parameters which are used in the proposed algorithm and the parameters of the randomly generated graphs are presented in Table 6 where the first column indicates the number of the subtasks in the randomly generated DAG, the second column shows the upper bound of the levels in a DAG, third column depicts the computation cost

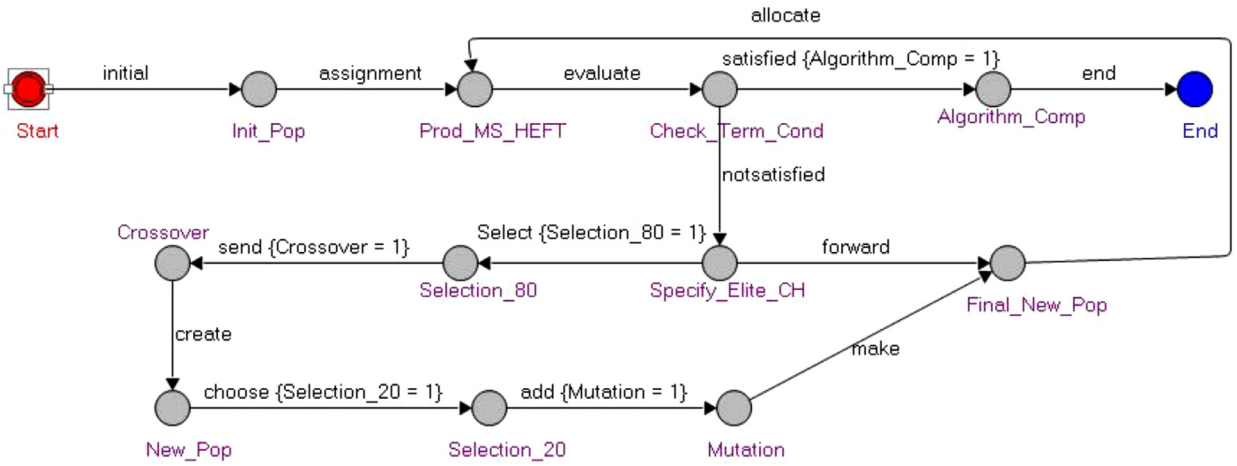


Fig. 12. The event-based model in PAT tool.

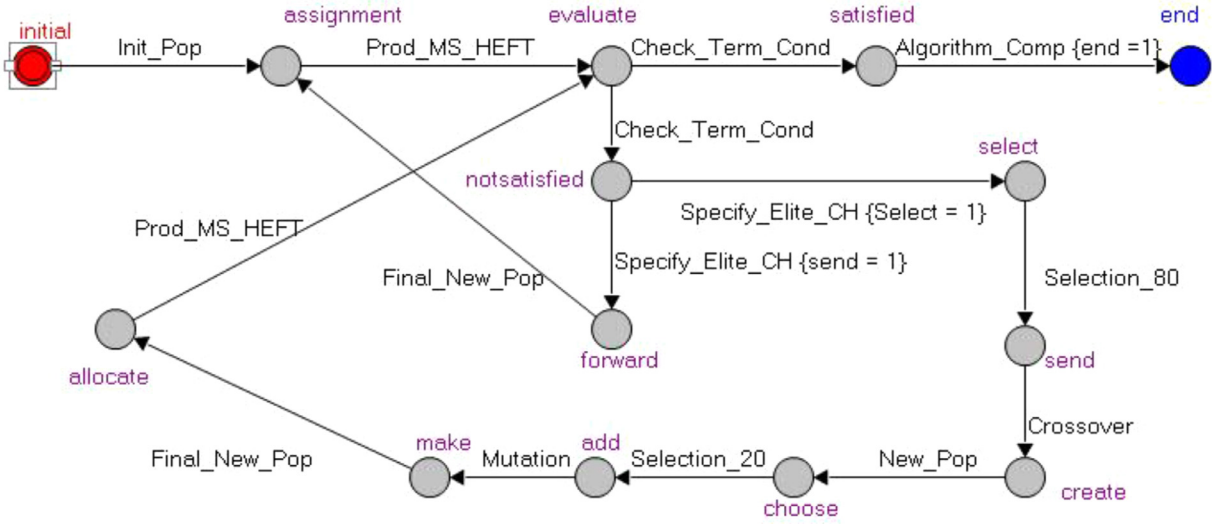


Fig. 13. The action based model in PAT tool.

Table 6
Parameters of the random task graphs.

Tasks	Maximum levels	Computation cost range	Communication cost range	Degree of subtasks
10	5	Randomly from (1–50)	Randomly from (1–40)	Randomly
20	9	Randomly from (1–50)	Randomly from (1–40)	Randomly
50	24	Randomly from (1–50)	Randomly from (1–40)	Randomly
100	49	Randomly from (1–50)	Randomly from (1–40)	Randomly

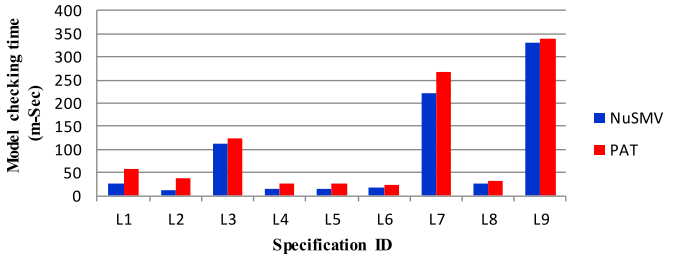


Fig. 14. The model checking time for each the specification in the event based model using NuSMV and PAT model checkers.

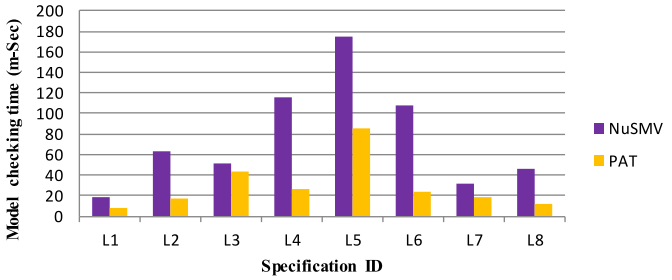


Fig. 15. The model checking time for each the specification in the action based model using NuSMV and PAT model checkers.

which selected randomly from a range (1–50), the fourth column shows the communication cost of the edge between two subtasks which selected randomly from a range (1–40) and eventually the last column represents the way of selection the number of succes-

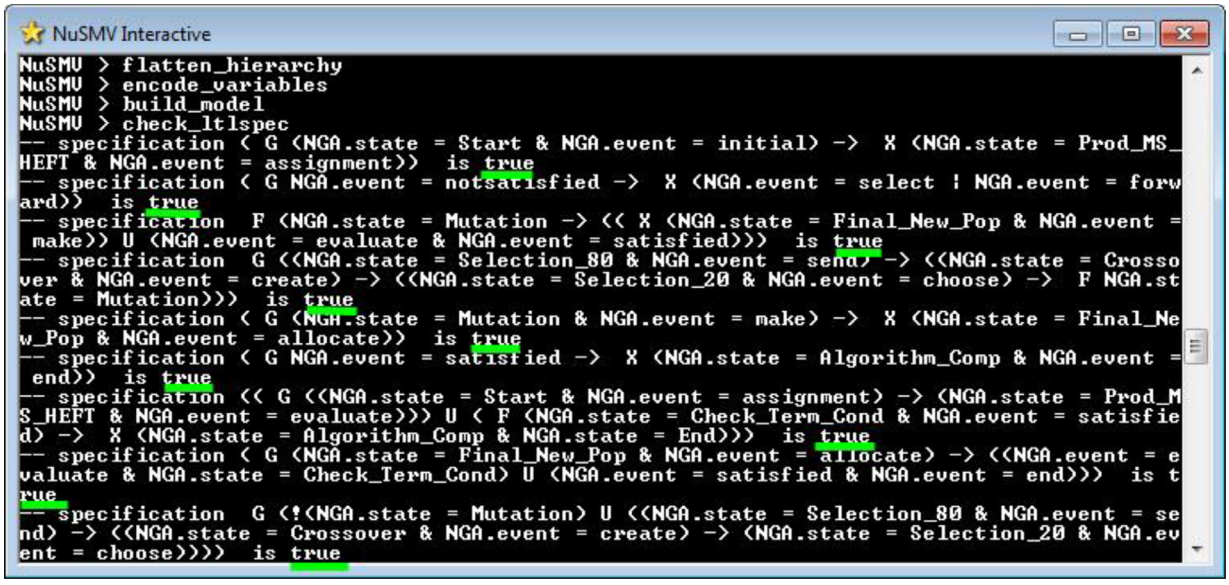


Fig. 16. The verification results of the event based model in NuSMV.

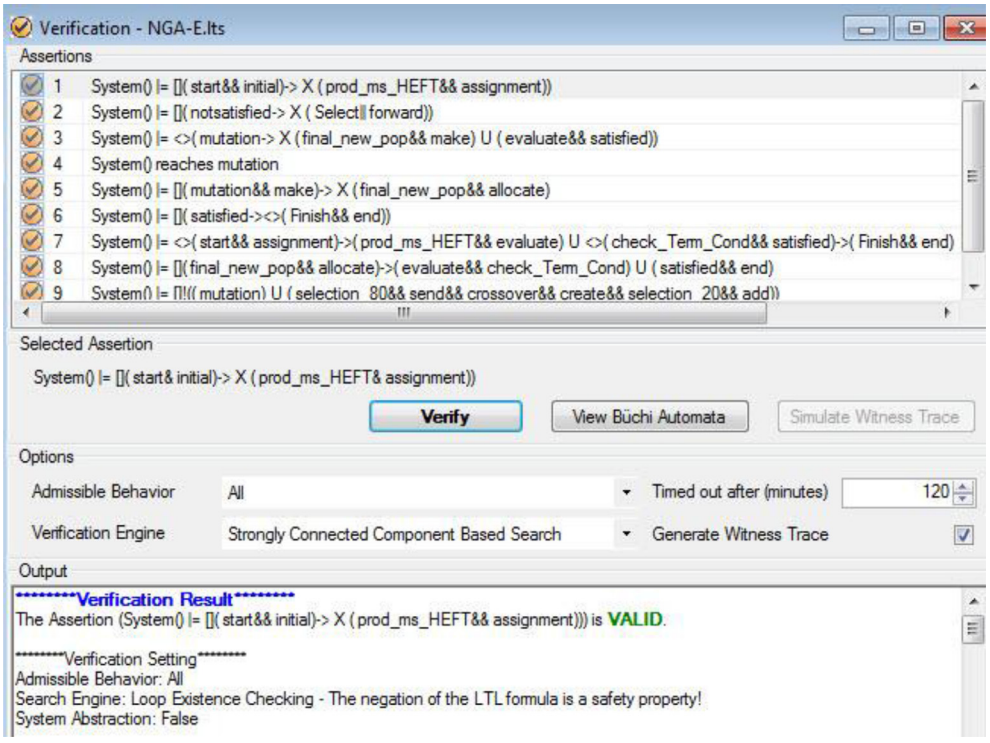


Fig. 17. The verification results of the event based model in PAT.

sors for the current task which depends on the tasks that can be the successor of the current task.

5.1. Performance metrics

Schedule Length Ratio (SLR) and efficiency metrics are used in this section to assess the performance of the N-GA algorithm with the three heuristics and MPQMA algorithms.

SLR: Schedule length is the main performance measure of a scheduling algorithm on a DAG. Since a large set of application graphs with different characteristic is used, it is necessary to normalize the schedule length of each graph to a lower bound. SLR is defined as the normalized schedule length to the lower bound of the schedule length. The SLR value of an algorithm for a task graph is obtained by Eq. (11):

$$SLR = \frac{makespan}{\sum_{t_i \in CP_{min}} \min_{p_k \in P} (W(t_i, p_k))} \tag{11}$$

Where, the CP_{min} is the critical path of the unscheduled application based on the computation cost of tasks on the fastest processor p_k (Xu et al., 2014) . The denominator is equal to the sum of computation costs of tasks located on CP_{min} when they are scheduled on p_k. The minimum of SLR value of any algorithm is 1, since the denominator in the Eq. (11) is the lower bound for the makespan

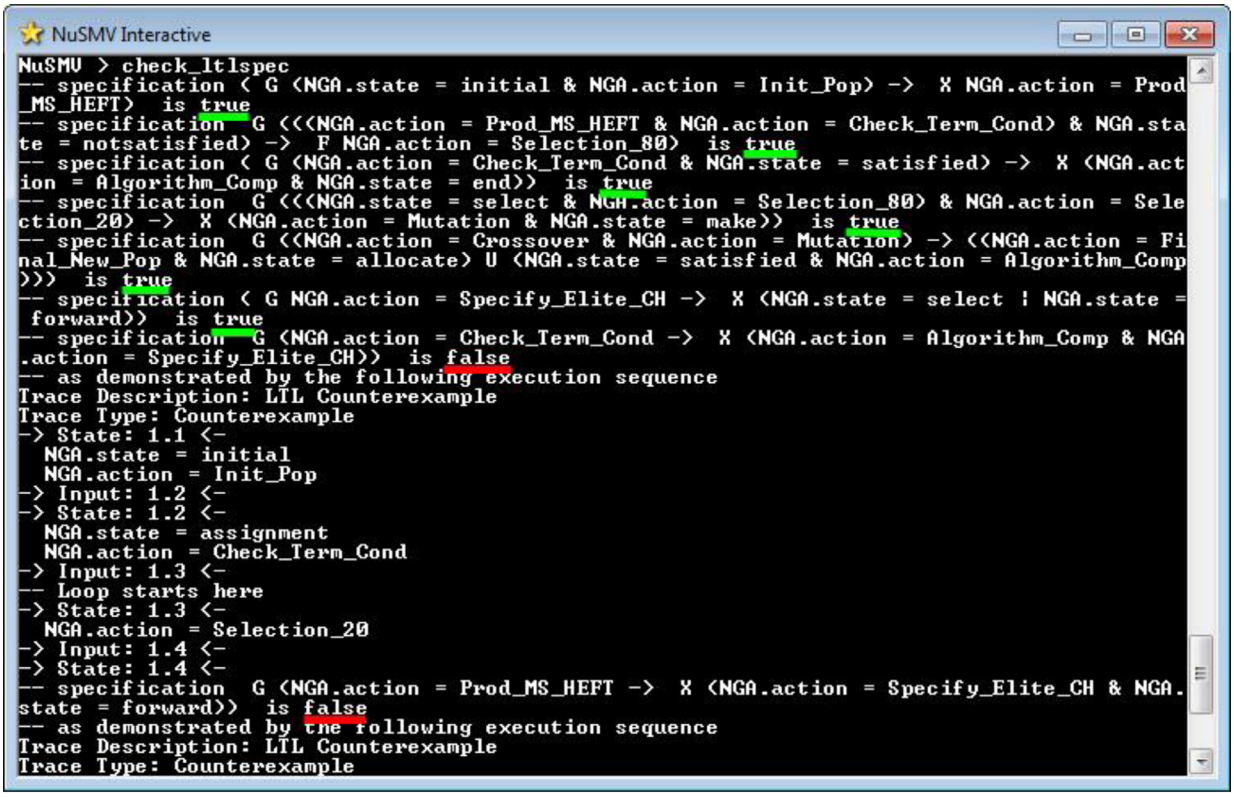


Fig. 18. The verification results of the action based model in NuSMV.

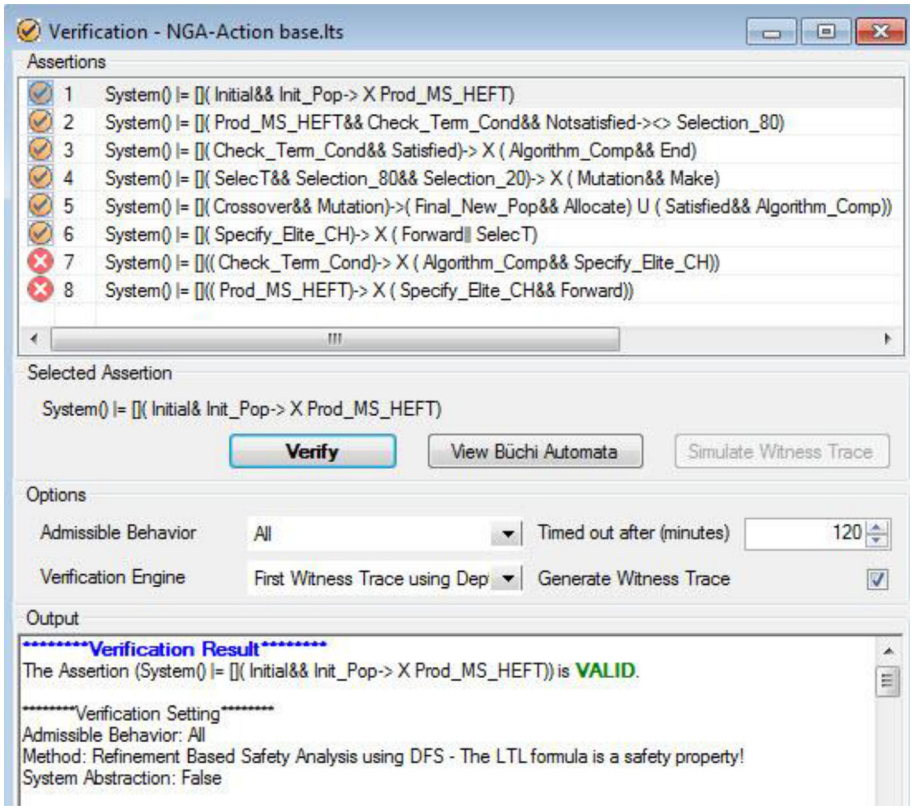


Fig. 19. The verification results of the action based model in PAT.

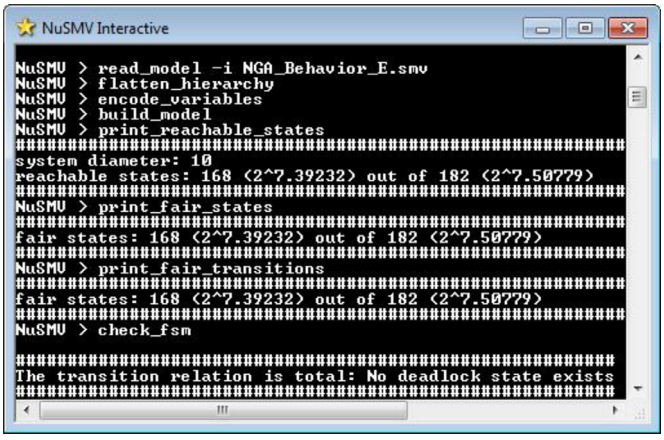


Fig. 20. The correctness of the event based model in NuSMV.

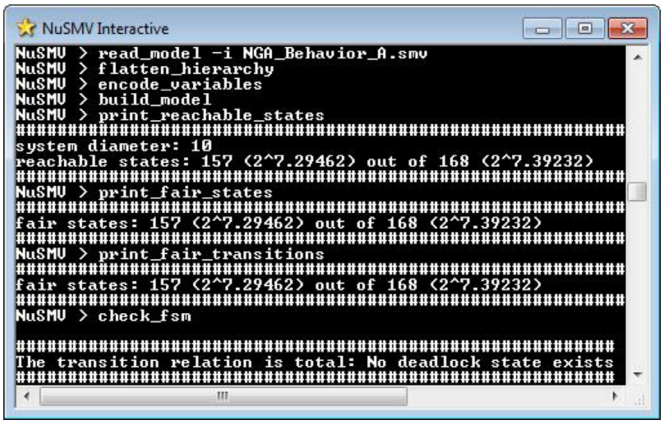


Fig. 21. The correctness of the action based model in NuSMV.

of the graph. Small value of SLR for comparing the performance of scheduling algorithms is better than the large one.

Communication to Computation cost Ratio (CCR): The average communication cost divided by the average computation cost of the application DAG. The CCR value of an algorithm for a task graph is obtained by Eq. (12):

$$CCR = \frac{\frac{1}{E}\sum_{\text{edge}(t_i,t_j)\in E} C(t_i, t_j)}{\frac{1}{T}\sum_{t_i \in T} \overline{W(t_i)}} \tag{12}$$

Where, E is the set of edges and T is the set of subtasks in the DAG, $C(t_i, t_j)$ is the communication cost of the edge from subtask t_i to the subtask t_j and $\overline{W(t_i)}$ is the average computational cost of subtask t_i.

Speedup: The speedup of a task schedule is the ratio of the serial schedule length obtained by assigning all tasks to the fastest processor, to the parallel execution time of the task schedule and speedup is obtained by Eq. (13):

$$Speedup_{min} = \frac{min_{p_k \in P}\left(\sum_{t_i \in T} W(t_i, p_k)\right)}{makespan} \tag{13}$$

Where, $W(t_i, p_k)$ is the cost of the subtask t_i on the processor p_k.

Efficiency: Efficiency is the ratio of the speedup value to the number of processors used which is obtained by Eq. (14):

$$Efficiency = \frac{Speedup_{\min}}{m} \tag{14}$$

Where, m is the number of processors in a heterogeneous computing system.

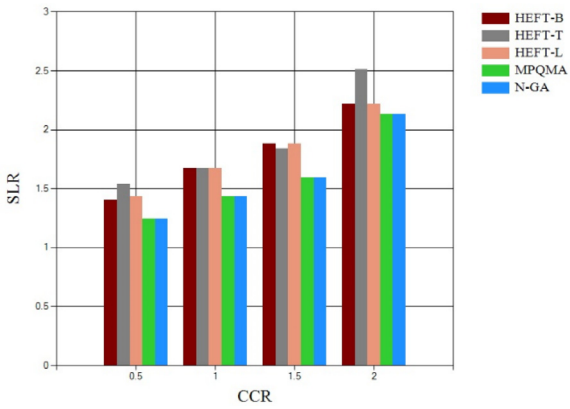


Fig. 22. SLR in different CCR for the DAG in Fig. 1.

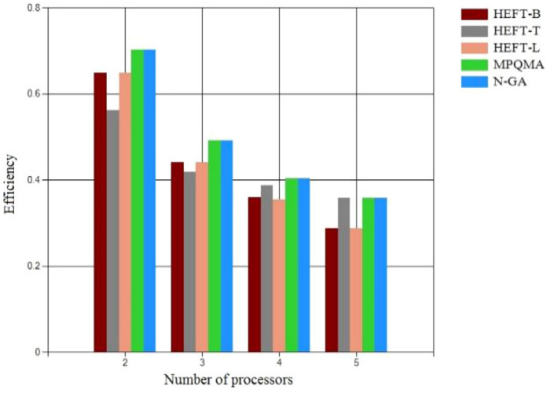


Fig. 23. Efficiency in the different number of processors for the DAG in Fig. 1.

The used CCR values in the experiments for the DAG in Fig. 1 are 0.5, 1.0, 1.5 and 2.0. Figs. 22 and 23 show the average SLR and the efficiency of algorithms under different CCR values and a different number of heterogeneous processors respectively. Fig. 20 indicates that the average SLR values of all algorithms are increased when the CCR value is increased. Fig. 23 shows the efficiency of the algorithms vs. the different numbers of processors. With regard to the results in Figs. 22 and 23, the N-GA and MPQMA algorithms have better performance than other three algorithms in terms of both SLR and efficiency.

5.2. Random generated application graphs and statistical analysis

Randomly generated DAGs help us to assess diverse application graphs. We have evaluated the performance of the proposed algorithm under different parameters, including different numbers of subtasks and different CCR values. The N-GA is compared with other algorithms in terms of the makespan. The number of generated subtasks in a DAG is selected from 10, 20, 50, and 100. The computational and communication (edge) costs of the DAG are generated randomly from a range according to the Table 6. Also in this section, Statistical approaches in SPSS tool are used to evaluate the proposed algorithm with the HETF-B, HEFT-T, HEFT-L and MPQMA algorithms.

5.2.1. Random graphs with 10 subtasks

Fig. 24 represents the makespans of 100 different randomly generated DAGs with 10 subtasks. According to the statistical results of the 100 graphs, the results in boxplot (Fig. 25) and Table 7 show that the N-GA outperforms HEFT-B, HEFT-T, and HEFT-L in terms of makespan as well as MPQMA, but the average values of

Table 7
The means of makespans and execution time for graphs in Figs. 24 and 26.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	102.8000	103.0400	101.7200	95.4600	95.4600
Execution time(mean(ms))	–	–	–	45.4	11.56

Table 8
The means of makespans and execution time for graphs in Figs. 27 and 29.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	187.8000	192.6600	187.8800	173.5400	173.7800
Execution time(mean(ms))	–	–	–	681	164

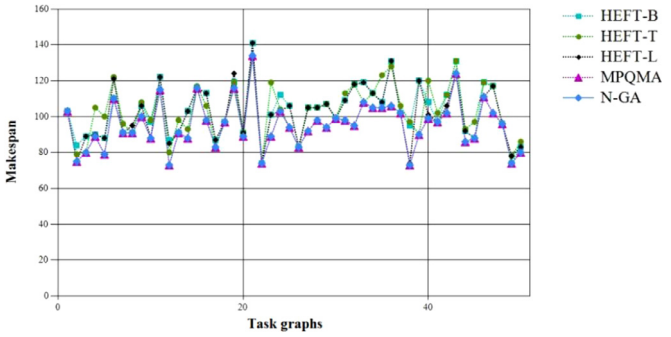


Fig. 24. The makespans of the 100 different randomly generated DAGs (subtasks = 10, the number of processors = 8 and PopSize = 40).

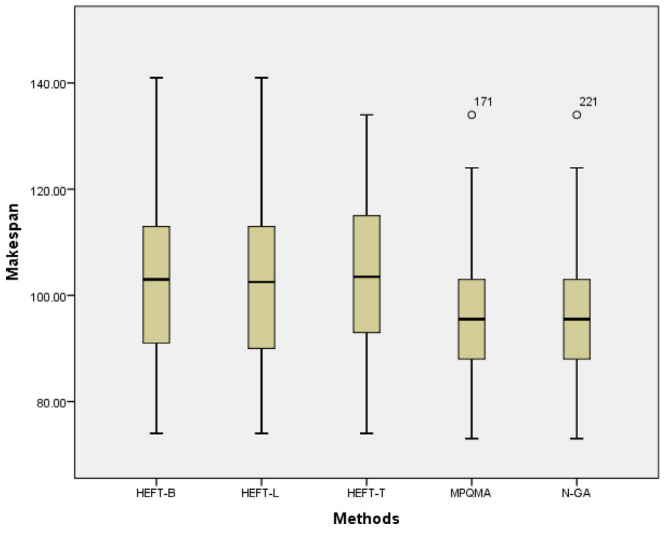


Fig. 25. Boxplot of makespans using different methods for the graphs in Fig. 24.

the execution time from Fig. 26 which presented in Table 7 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA.

5.2.2. Random graphs with 20 subtasks

Fig. 27 shows the makespans of 100 different randomly generated DAGs with 20 subtasks. According to the statistical results of the 100 graphs, the results in boxplot (Fig. 28) and Table 8 show that the N-GA outperforms HEFT-B, HEFT-T, and HEFT-L in terms of makespan as well as the MPQMA, but the average values of the execution time from Fig. 29 which presented in Table 8 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA.

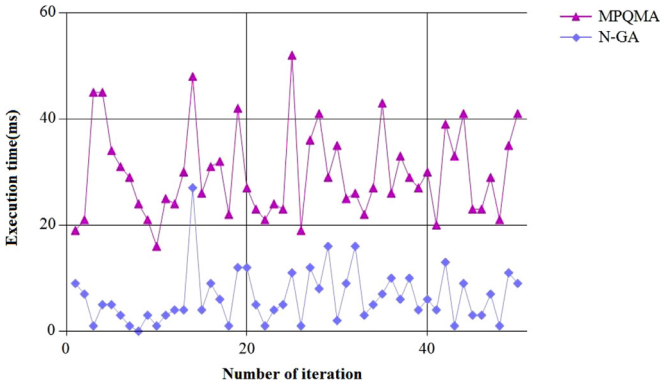


Fig. 26. The execution time of N-GA and MPQMA methods for graphs in Fig. 24.

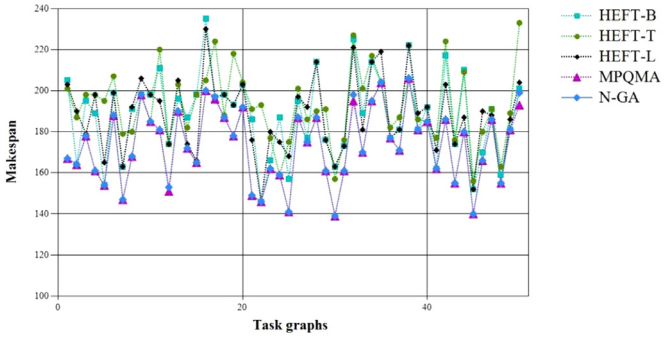


Fig. 27. The makespans of the 100 different randomly generated DAGs (subtasks = 20, the number of processors = 8 and PopSize = 80).

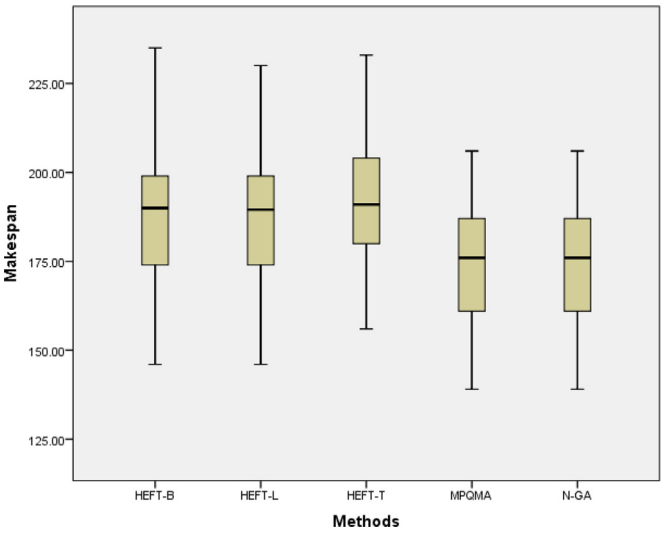


Fig. 28. Boxplot of makespans using different methods for the graphs in Fig. 27.

Table 9
The means of makespans and execution time for graphs in Figs. 30 and 32.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(Mean)	469.7000	479.4400	467.7000	421.5600	424.2000
Execution Time(Mean(ms))	–	–	–	34132.38	12699.04

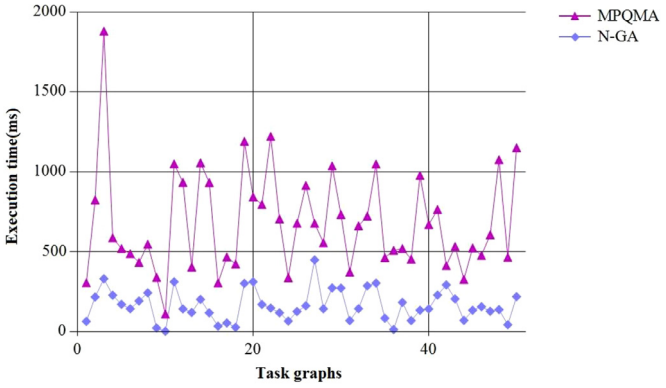


Fig. 29. The execution time of N-GA and MPQMA methods for graphs in Fig. 27.

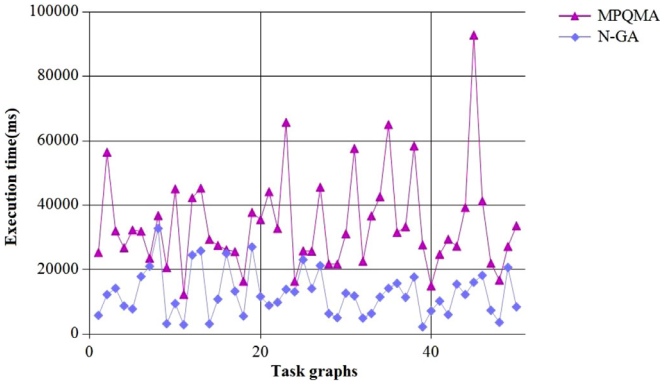


Fig. 32. The execution time of N-GA and MPQMA methods for graphs in Fig. 30.

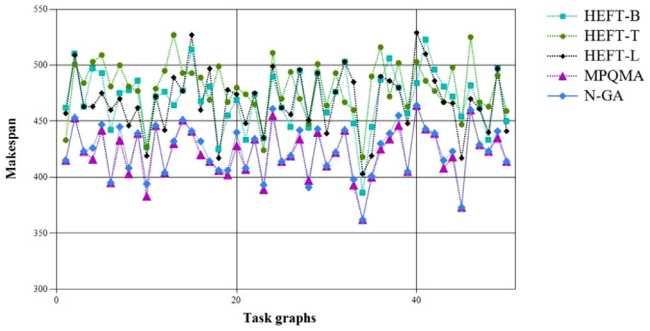


Fig. 30. The makespans of the 100 different randomly generated DAGs (subtasks = 50, the number of processors = 8 and PopSize = 200).

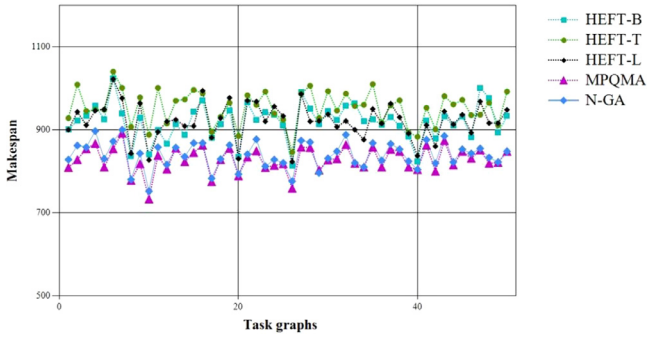


Fig. 33. The makespans of the 100 different randomly generated DAGs (subtasks = 100, the number of processors = 8 and PopSize = 400).

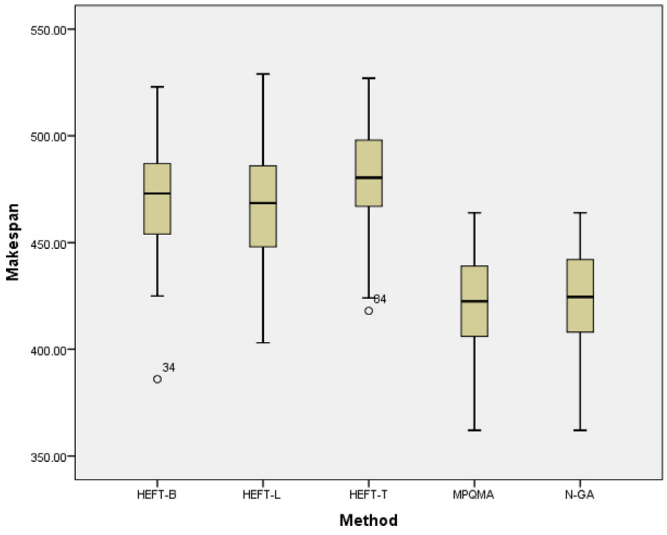


Fig. 31. Boxplot of makespans using different methods for the graphs in Fig. 30.

5.2.3. Random graphs with 50 subtasks

Fig. 30 shows the makespans of 100 different randomly generated DAGs with 50 subtasks. According to the statistical results of the 100 graphs, the results in boxplot (Fig. 31) and Table 9 show that the N-GA outperforms HEFT-B, HEFT-T, and HEFT-L in terms of makespan as well as MPQMA but the average values of the execution time from Fig. 32 which presented in Table 9 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA.

5.2.4. Random graphs with 100 subtasks

Fig. 33 shows the makespans of 100 different randomly generated DAGs with 100 subtasks. According to the statistical results of the 100 graphs, the results in boxplot (Fig. 34) and Table 10 show that the N-GA outperforms HEFT-B, HEFT-T, and HEFT-L in terms of makespan as well as MPQMA, but the average values of the execution time from Fig. 35 which presented in Table 10 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA.

5.3. Convergence trace

In this section, the N-GA and MPQMA solutions convergence behavior on the randomly generated DAGs are examined and other three heuristic algorithms are ignored because of their nature which they have only one solution in every execution. Fig. 36 shows the 30 independent runs of the N-GA and MPQMA for a randomly generated DAG with 10 subtasks. The average values of the makespans and execution time in all the 30 repetitions from Figs. 36 and 37 presented in Table 11 reveal that the proposed al-

Table 10
The means of makespans and execution time for graphs in Figs. 33 and 35.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	919.3800	953.2000	921.5400	828.4800	839.6000
Execution time(mean(ms))	–	–	–	838755.66	314317.28

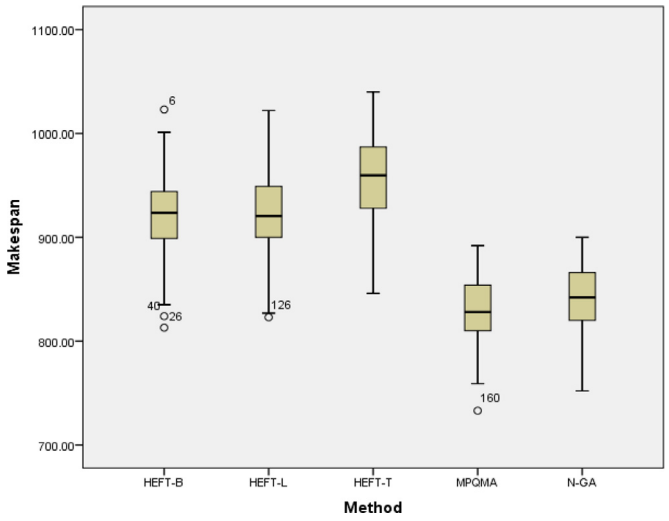


Fig. 34. Boxplot of makespans using different methods for the graphs in Fig. 33.

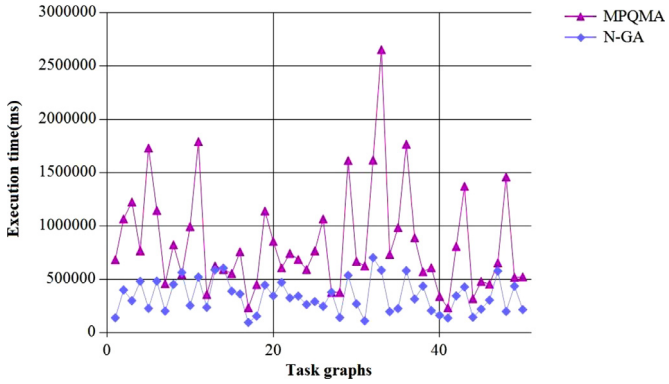


Fig. 35. The execution time of N-GA and MPQMA methods for graphs in Fig. 33.

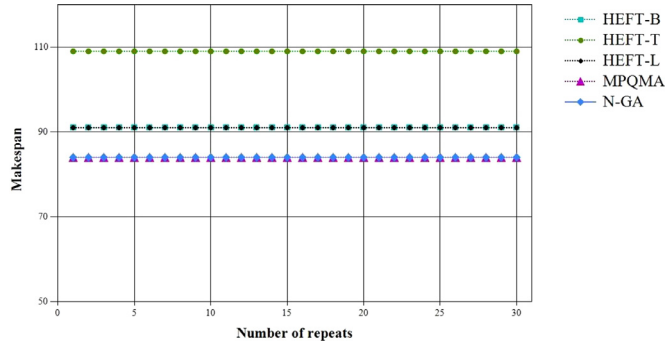


Fig. 36. The 30 independent runs of the N-GA and MPQMA for a randomly generated DAG (subtasks = 10, processor = 8, PopSize = 40).

Table 11
The means of makespans and execution time for graphs in Figs. 36 and 37.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	91	109	91	84	84
Execution time(mean(ms))	–	–	–	28.833	4

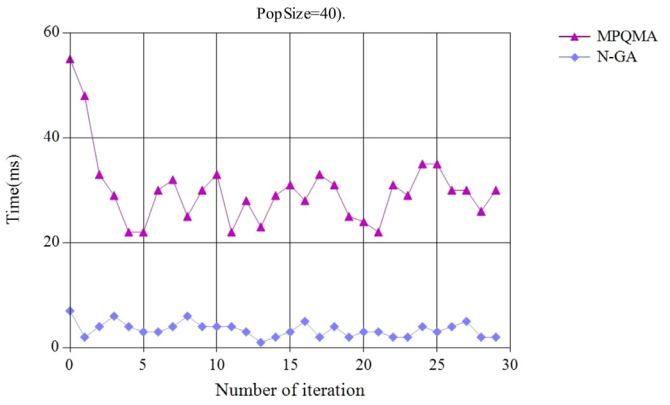


Fig. 37. The execution time of N-GA and MPQMA methods for graphs in Fig. 36.

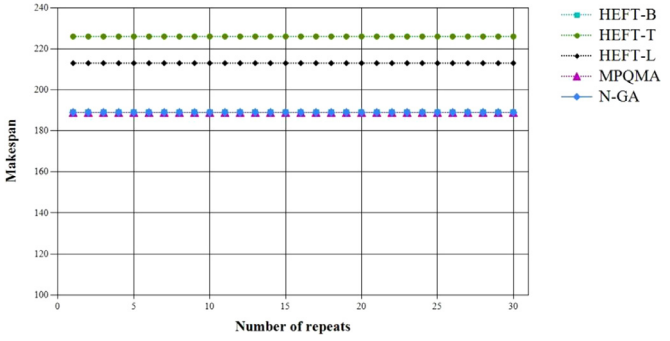


Fig. 38. The 30 independent run of the N-GA and MPQMA for a randomly generated DAG (subtasks = 20, processors = 8, PopSize = 80).

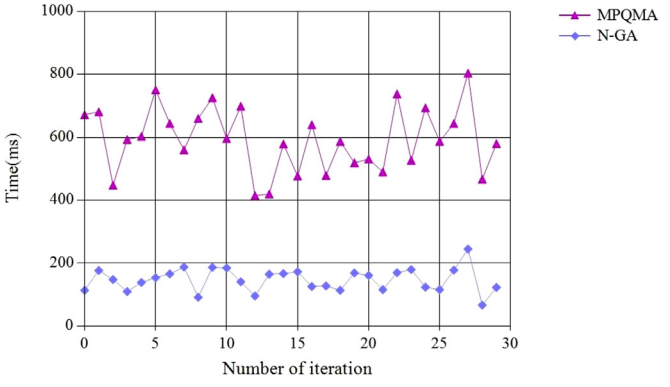


Fig. 39. The execution time of N-GA and MPQMA methods for graphs in Fig. 38.

gorithm significantly decreases the execution time of the MPQMA while they obtain the same makespan.

Fig. 38 shows 30 independent runs of the N-GA and MPQMA for a randomly generated DAG with 20 subtasks. The average values of the makespans and execution time in all the 30 repetitions from Figs. 38 and 39 presented in Table 12 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA while they obtain the same makespan.

Table 12
The means of makespans and execution time for graphs in Figs. 38 and 39.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	189	226	213	189	189
Execution time(mean(ms))	–	–	–	592.933	147.8

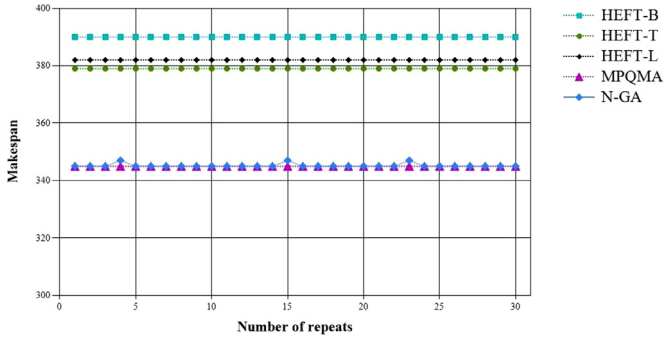


Fig. 40. The 30 independent run of the N-GA and MPQMA for a randomly generated DAG (subtasks = 50, processors = 8, PopSize = 200).

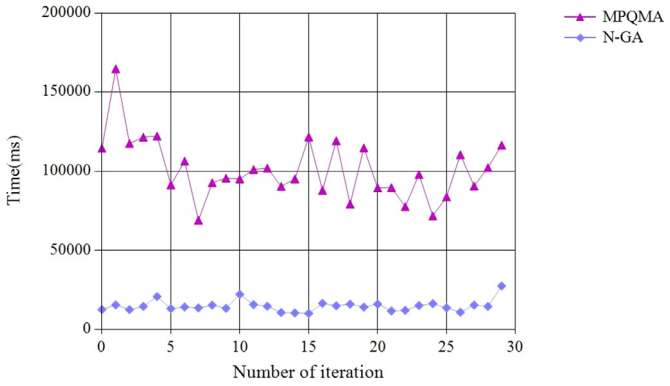


Fig. 41. The execution time of N-GA and MPQMA methods for graphs in Fig. 40.

Table 13
The means of makespans and execution time for graphs in Figs. 40 and 41.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	390	379	382	345	345.5
Execution time(mean(ms))	–	–	–	101126.866	14847.6

Table 14
The means of makespans and execution time for graphs in Figs. 42 and 43.

	HEFT-B	HEFT-T	HEFT-L	MPQMA	N-GA
Makespan(mean)	695	686	645	606.5667	616.7667
Execution time(mean(ms))	–	–	–	4302006.266	567019.6

Fig. 40 shows 30 independent runs of the N-GA and MPQMA for a randomly generated DAG with 50 subtasks. The average values of the makespans and execution time in all the 30 repetitions from Figs. 40 and 41 presented in Table 13 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA while the N-GA obtain the close makespan of the MPQMA.

Fig. 42 shows 30 independent runs of the N-GA and MPQMA for a randomly generated DAG with 10 subtasks. The average values of the makespans and execution time in all the 30 repetitions from Figs. 42 and 43 which presented in Table 14 reveal that the proposed algorithm significantly decreases the execution time of the MPQMA while the N-GA obtain the close makespan of the MPQMA.

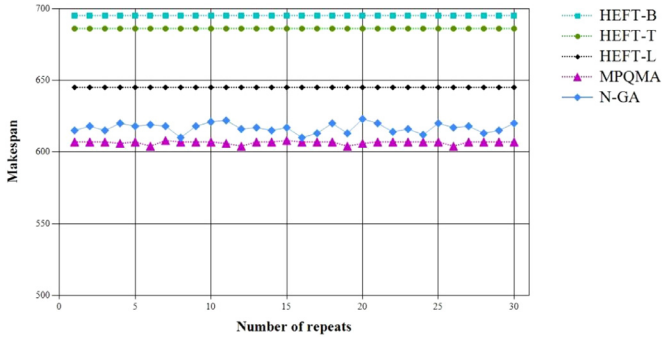


Fig. 42. The 30 independent run of the N-GA and MPQMA for a randomly generated DAG (subtasks = 100, processor = 8, PopSize = 400).

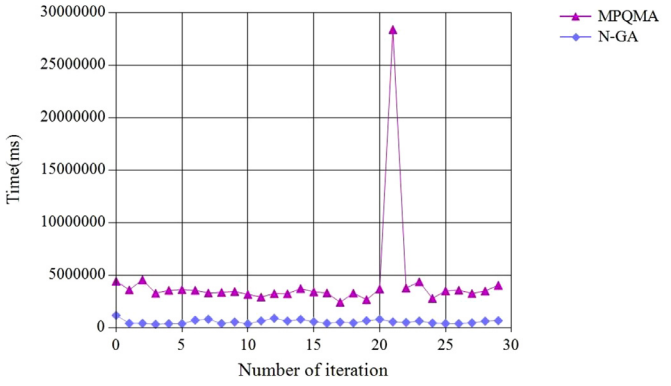


Fig. 43. The execution time of N-GA and MPQMA methods for graphs in Fig. 42.

6. Conclusions and future work

In this paper, an improved genetic algorithm for static task scheduling in the cloud environment is proposed which is called N-GA. This algorithm uses GA along with a heuristic-based HEFT search to assign subtasks to processors. Experiment and statistical analysis results indicated that our proposed algorithm outperforms HEFT-B, HEFT-T, and HEFT-L algorithms' makespan and also optimized the execution time of our recent memetic algorithm significantly. In addition, we presented a behavioral model for analyzing the correctness of the proposed algorithm's behavior. We extracted the expected specifications of the N-GA algorithm in the form of Linear Temporal Logic (LTL) formulas. By using two model checking mechanisms (event based and action based labeled transition system), we verified the proposed behavioral model using NuSMV and PAT model checkers. Then, we analyzed the correctness of the proposed N-GA algorithm in terms of expected specifications of the proposed behavioral model. The verification results show that the behavioral model supports some logical problems such as reachability, fairness, and deadlock-free. Also, we compared the performance of NuSMV and PAT model checkers to complete the verification workload. According to the statistic schema of model checking for each the specification, the model checking time of NuSMV model checker is lower than the model checking the time of PAT model checker in the event-based method. Also, the model checking time of the PAT model checker is lower than the time of the NuSMV model checker in the action based method. In the future work, we will use other evolutionary algorithms for scheduling problem in the cloud environment. Furthermore, we will parallelize the proposed algorithm to further decrease the time needed to find solutions.

References

Abed, I., et al., 2014. Optimization of the time of task scheduling for dual manipulators using a modified electromagnetism-like algorithm and genetic algorithm. Arabian J. Sci. Eng 39 (8), 6269–6285.

Abo-Hammour, Z.S., Alsmadi, O.M.K., Al-Smadi, A.M., 2011. Multi-time-scale systems model order reduction via genetic algorithms with eigenvalue preservation. J. Circ. Syst. Comput 20 (07), 1403–1418.

Alkhanak, E.N., Lee, S.P., Khan, S.U.R., 2015. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. Future Gen. Comput. Syst 50, 3–21.

Al-Saqqar, F., et al., 2015. Model checking temporal knowledge and commitments in multi-agent systems using reduction. Simul. Model. Pract. Theory 51, 45–68.

Al-Saqqar, F., Bentahar, J., Sultan, K., 2016. On the soundness, completeness and applicability of the logic of knowledge and communicative commitments in multi-agent systems. Expert Syst. Appl 43, 223–236.

Ashouraie, M., Jafari Navimipour, N., 2015. Priority-based task scheduling on heterogeneous resources in the Expert Cloud. Kybernetes 44 (10), 1455–1471.

Bae, K., Meseguer, J., 2015. Model checking linear temporal logic of rewriting formulas under localized fairness. Sci. Comput. Programm 99, 193–234.

Baier, C., Katoen, J.-P., 2008. Principles of Model Checking (Representation and Mind Series). The MIT Press, p. 975.

Bentahar, J., et al., 2013. Symbolic model checking composite Web services using operational and control behaviors. Expert Syst. Appl 40 (2), 508–522.

Charband, Y., Navimipour, N.J., 2016. Online knowledge sharing mechanisms: A systematic review of the state of the art literature and recommendations for future research. Inf. Syst. Front 1–21.

Chiregi, M., Jafari Navimipour, N., 2016. Trusted services identification in the cloud environment using the topological metrics. Karbala Int. J. Mod. Sci.

Chiregi, M., Navimipour, N.J., 2016. A new method for trust and reputation evaluation in the cloud environments using the recommendations of opinion leaders' entities and removing the effect of troll entities. Comput. Hum. Behav 60, 280–292.

Clarke, E.M., Grumberg, O., Peled, D.A., 1999. Model Checking. MIT press.

Clavel, M., et al., 2007. LTL model checking. In: All About Maude - A High-Performance Logical Framework. Springer, Berlin Heidelberg, pp. 385–418.

Daoud, M.I., Kharma, N., 2011. A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. J. Parallel Distrib. Comput 71 (11), 1518–1531.

El Kholy, W., et al., 2014. Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols. Expert Syst. Appl 41 (16), 7478–7494.

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.

Habibizad Navin, A., et al., 2014. Expert grid: New type of grid to manage the human resources and study the effectiveness of its task scheduler. Arabian J. Sci. Eng 39 (8), 6175–6188.

Holland, J.H., 1975. Adaptation in Natural And Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press, Ann Arbor.

Huang, S.-Y., Cheng, K.-T., 1998. Formal Equivalence Checking and Design DeBugging. Kluwer Academic Publishers, p. 229.

Jafari Navimipour, N., et al., 2014. Job scheduling in the Expert Cloud based on genetic algorithms. Kybernetes 43 (8), 1262–1275.

Jafari Navimipour, N., et al., 2015. Behavioral modeling and automated verification of a Cloud-based framework to share the knowledge and skills of human resources. Comput. Ind 68, 65–77.

Jafari Navimipour, N., et al., 2015. Expert Cloud: A Cloud-based framework to share the knowledge and skills of human resources. Comput. Hum. Behav 46 (C), 57–74.

Jafari Navimipour, N., Charband, Y., 2016. Knowledge sharing mechanisms and techniques in project teams: literature review, classification, and current trends. Comput. Hum. Behav.

Jeffery, R., et al., 2015. An empirical research agenda for understanding formal methods productivity. Inf. Softw. Technol 60, 102–112.

Jun, D., et al., 2013. EasyHPS: a multilevel hybrid parallel system for dynamic programming. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International.

Kenli, L., et al., 2014. A DAG task scheduling scheme on heterogeneous computing systems using invasive weed optimization algorithm. Parallel Architectures, Algorithms and Programming (PAAP), 2014 Sixth International Symposium on.

Keshanchi, B., Jafari Navimipour, N., 2015. Priority-based task scheduling on cloud computing environment using a memetic algorithm. J. Circ. Syst. Comput.

Khan, M.A., 2012. Scheduling for heterogeneous systems using constrained critical paths. Parallel Comput 38 (4–5), 175–193.

Loveland, D.W., 1978. Automated Theorem Proving: A Logical Basis (Fundamental Studies in Computer Science). Elsevier North-Holland sole distributor for the U.S.A. and Canada.

Malawski, M., et al., 2015. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Future Gen. Comput. Syst 48, 1–18.

Milani, B.A., Navimipour, N.J., 2016. A comprehensive review of the data replication techniques in the cloud environments: major trends and future directions. J. Netw. Comput. Appl 64, 229–238.

Mirjalili, S., Mirjalili, S.M., Lewis, A., 2014. Grey wolf optimizer. Adv. Eng. Softw 69, 46–61.

Nasiri, M.M., 2015. A modified ABC algorithm for the stage shop scheduling problem. Appl. Soft Comput 28 (0), 81–89.

Navimipour, N.J., 2015. A formal approach for the specification and verification of a trustworthy human resource discovery mechanism in the Expert Cloud. Expert Syst. Appl 42 (15), 6112–6131.

Navimipour, N.J., Khanli, L.M., 2008. The LGR method for task scheduling in computational grid. Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on.

Navimipour, N.J., Rahmani, A.M., 2009. The new genetic based method with optimum number of super node in heterogeneous wireless sensor network for fault tolerant system. Intelligent Networking and Collaborative Systems, 2009. INCOS '09. International Conference on.

Navimipour, N.J., Soltani, Z., 2016. The impact of cost, technology acceptance and employees' satisfaction on the effectiveness of the electronic customer relationship management systems. Comput. Hum. Behav 55, 1052–1066.

Navimipour, N.J., Zareie, B., 2015. A model for assessing the impact of e-learning systems on employees' satisfaction. Comput. Hum. Behav 53, 475–485.

Navin, A.H., et al., 2014. Expert grid: new type of grid to manage the human resources and study the effectiveness of its task scheduler. Arabian J. Sci. Eng 39 (8), 6175–6188.

Ong, B., Fukushima, M., 2011. Genetic algorithm with automatic termination and search space rotation. Memetic Comput 3 (2), 111–127.

Ouchani, S., Debbabi, M., 2015. Specification, verification, and quantification of security in model-based systems. Computing 97 (7), 691–711.

Panda, S., Jana, P., 2015. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput 71 (4), 1505–1533.

Rezaee, A., et al., 2014. Formal process algebraic modeling, verification, and analysis of an abstract fuzzy inference cloud service. J. Supercomput 67 (2), 345–383.

Rozier, K.Y., 2011. Linear temporal logic symbolic model checking. Comput. Sci. Rev 5 (2), 163–203.

Safarkhanlou, A., et al., 2015. Formalizing and verification of an antivirus protection service using model checking. Procedia Comput. Sci 57, 1324–1331.

Sinnen, O., To, A., Kaur, M., 2011. Contention-aware scheduling with task duplication. J. Parallel Distrib. Comput 71 (1), 77–86.

Soltani, Z., Navimipour, N.J., 2016. Customer relationship management mechanisms: A systematic review of the state of the art literature and recommendations for future research. Comput. Hum. Behav 61, 667–688.

Souri, A., Jafari Navimipour, N., 2014. Behavioral modeling and formal verification of a resource discovery approach in grid computing. Expert Syst. Appl 41 (8), 3831–3849.

Souri, A., Norouzi, M., 2015. A new probable decision making approach for verification of probabilistic real-time systems. Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on.

Souri, A., Sharifloo, M.A., Norouzi, M., 2012. Analyzing SMV & UPPAAL model checkers in real-time systems. Global J. Technol 1.

Tang, Z., et al., 2015. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. J. Grid Comput 1–20.

Topcuoglu, H., Hariri, S., Min-You, W., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. Parallel Distrib. Syst. IEEE Trans 13 (3), 260–274.

Tripathy, B., Dash, S., Padhy, S.K., 2015. Dynamic task scheduling using a directed neural network. J. Parallel Distrib. Comput 75 (0), 101–106.

Ucar, B., et al., 2006. Task assignment in heterogeneous computing systems. J. Parallel Distrib. Comput 66 (1), 32–46.

Ullman, J.D., 1975. NP-complete scheduling problems. J. Comput. Syst. Sci 10 (3), 384–393.

Xu, Y., et al., 2013. A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization. J. Parallel Distrib. Comput 73 (9), 1306–1322.

Xu, Y., et al., 2014. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf. Sci 270 (0), 255–287.

Yeung, W.L., 2011. Behavioral modeling and verification of multi-agent systems for manufacturing control. Expert Syst. Appl 38 (11), 13555–13562.

Zareie, B., Jafari Navimipour, N., 2016. The effect of electronic learning systems on the employee's commitment. Int. J. Manage. Edu 14 (2), 167–175.

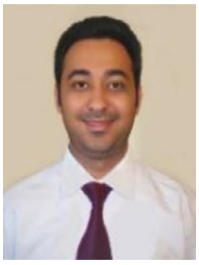
Zareie, B., Navimipour, N.J., 2016. The impact of electronic environmental knowledge on the environmental behaviors of people. Comput. Hum. Behav 59, 1–8.

Zeng, L., Veeravalli, B., Li, X., 2015. SABA: a security-aware and budget-aware workflow scheduling strategy in clouds. J. Parallel Distrib. Comput 75, 141–151.

Zhang, B., et al., 2014. A verification framework with application to a propulsion system. Expert Syst. Appl 41 (13), 5669–5679.

Zhao, Y., Rozier, K.Y., 2014. Formal specification and verification of a coordination protocol for an automated air traffic control system. Sci. Comput. Programm 96, 337–353 **Part 3**.

Bahman Keshanchi received his B.S. degree in Software Engineering from Bostan-Abad Branch, Islamic Azad University, Bostan-Abad, Iran, in 2011 and his M.Sc. degree in Software Engineering from Science and Research Branch, Islamic Azad University, Tabriz, Iran in 2015. His research interests include Grid & Cloud computing, Task Scheduling, programming.



Alireza Souri received his B.S. degree in Software Engineering from University College of Nabi Akram, Iran, in 2011 and his M.Sc. degree in Software Engineering from Science and Research Branch, Islamic Azad University, Iran in 2013. Currently, he is a researcher and lecturer at Islamic Azad University. Up to now, he has authored/co-authored 12 academic articles. He served on the program committees and the technical reviewer of several international conferences and journals. His research interests include Formal Specification & Verification, Model checking, Grid & Cloud computing. Now, He is a member of The Society of Digital Information and Wireless Communications.

Nima Jafari Navimipour received his B.S. in computer engineering, software engineering, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2007; the M.S. in computer engineering, computer architecture, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2009; the Ph.D. in computer engineering, computer architecture, from Science and Research Branch, Islamic Azad University, Tehran, Iran in 2014. He is an assistance professor in the Department of Computer Engineering at Tabriz Branch, Islamic Azad University (the world's third largest university), Tabriz, Iran. He has published more than 60 papers in various journals and conference proceedings. His research interests include Cloud Computing, Social Networks, Fault-Tolerance Software, Knowledge Management, Evolutionary Computing, and Formal Verification.