Chapter 5
Security-Aware Independent Batch Scheduling in Computational Grids

Abstract. This chapter presents a model for independent batch scheduling in Computational Grid that enables the aggregation of security requirements as additional scheduling criteria. Artificial Neural Network (ANN) module is an important component of this model. It is designed for supporting the security-aware evolutionary single- and multi-population grid schedulers.Based on a preliminary analysis of the trust levels of resources and security demand parameters of tasks, the neural network monitors the scheduling and task execution processes and generates the tasks-machines mapping "suggestions" based on the information about resource failures and the resulting tasks and machines characteristics. This information is used by the schedulers for an effective minimization of the scheduling objective function and the improvement of the system throughput.

5.1 Introduction

While the maximization of the resource utilization and profits of the resource owners are the key objectives of the grid scheduling, they may conflict with grid users' security requirements and system reliability . A major hurdle in effective job outsourcing in grid is caused by network security threats. The grid resources may not be accessible if the grid cluster is under attack. The system infections may lead to machine crashes during the execution of tasks dispatched to that cluster. Therefore, it is desirable to have a prior knowledge about the security demands from grid jobs and the trust level assured by a resource provider at the grid cluster. An effective grid scheduler must be then security-driven and resilient in response to all scheduling and risky conditions. It means that to achieve the successful tasks executions according the specified users' requirements, the relation between the assurance of secure computing services by a grid site or by a cluster node (security) and the behavior of a resource node (trust) must be defined and analyzed.

The main problem addressed in this chapter is an improvement of the effectiveness of the single- and multi-population genetic-based grid schedulers in the low-cost resource allocations under security constraints. The security awareness of those

J. Kołodziej: Evolutionary Hierarchical Multi-Criteria Metaheuristics, SCI 419, pp. 81–111.
springerlink.com

schedulers is supported by an *Artificial Neural Network (ANN)* module integrated with the system. Based on a prior analysis of trust levels of the resources and security demand parameters of tasks, the neural network monitors the scheduling and task execution, and produces task-machine mapping "suggestions" (recommendations) by using the system information, such as resource failure rates and system input parameters. Thereafter, based on the ANN "suggestions", sub-optimal schedules are generated and used in the initialization procedures of genetic-based schedulers for optimizing the main scheduling objective functions such as makespan and flowtime.

Despite the generation of the sub-optimal solution to the specified scheduling problems, the ANN module is *not considered* in this work as additional scheduler. It works in a "background" of the main scheduling process and monitors the scheduling results. However, the schedules generated by ANN may be accepted as the optimal solutions if the employed schedulers cannot generate the better ones.

According to the notation introduced in Sec. 1.4.2 the independent batch security-aware scheduling in which the makespan and flowtime are optimized in a hierarchical mode can be specified as follows:

$$Rm\left[\{b, indep, (stat, dyn), hier\}\right](C_{max}(sec)[C_{max}(ris)], F(sec)[F(risk)])) \quad (5.1)$$

where:

- $C_{max}(sec)$ – stands for a makespan as the primary scheduling objective under security constraints;
- $F(sec)$ – stands for a flowtime as the second scheduling objective under security constraints;
- $C_{max}(risk)$ – stands for a makespan as the primary scheduling objective in the risky scheduling mode;
- $F(risk)$ – stands for a flowtime as the second scheduling objective in the risky scheduling mode.

The procedures of calculating the $C_{max}(sec)$, $F(sec)$, $C_{max}(risk)$ and $F(risk)$ values will be defined later on. The interpretations of the remaining parameters are the same as in Sec. 2.1 (Eq. (2.1)).

This chapter extends the model and results presented in [18] by the implementation and the comparative analysis of the effectiveness of multi-population and single-population GA-based grid schedulers and the integration of the ANN module with *Sim-G-Batch* grid simulator. In the ANN module the *Minimal Completion Time (MCT)* algorithm is used for the generation of sub-optimal schedules.

5.2 Related Work

There has been a number of studies over the last years in which the security procedures in grid scheduling are verified in risky environments, where the resource trust parameters must be analyzed. The security-aware scheduling process in grid environment [65], [152], [165] is more difficult for the management than conventional scheduling defined for supercomputers, real-time, and parallel computers [66], [91],

[100]. Unfortunately, well-known scheduling approaches for grid computing largely ignore this security factor, with only a handful of exceptions.

A simple classification of security-aware grid models for an immediate job execution mode is presented by Humphrey and Thompson in [65]. They define a job control system for accessing grid information services through authentication. However, they did not elaborate on how a scheduler should be designed to address the security concerns in collaborative computing over distributed cluster environment. An extensive survey of the research endeavors in this domain is presented in [34].

Hwang et al. [67] developed an interesting fault-tolerance mechanism in CGs with a failure detection service, that enables the detection of both task failures and user's secure requirements in a dynamic environment. Abawajy [1] developed a model, that faces the system dynamics by a replication of the users' jobs at multiple grid sites in order to improve reliability of grid resources, and successful job executions.

Due to their high scalability heuristic methods seem to be the effective tools in solving the large-scale grid scheduling problem with additional security and resource reliability criteria [137]). However, security and task abortion mechanism are usually applied as the external procedures separated from the core of the scheduling system. For example, security requirements can be specified in the grid system by using a simple trust model [8].

Some recent security-aware approaches in CG scheduling are based on the game-theoretical models . In [136] and [137] the authors define the risky and secure conditions in online scheduling in CGs caused by software vulnerability and distrusted security policy. They apply the game model introduced in [92] for simulating the resource owners selfish behavior. The results presented in [137] are extended by Wu et al. in [155]. The authors consider the heterogeneity of fault-tolerance mechanism in a security-assured grid job scheduling and define four types of GA-based online schedulers for the simulation of fault-tolerance mechanisms.

In the aforementioned models the final decisions on the secure allocation of task to resources are made by the CG users who do not cooperate with each other. The costs of the risk-resilient tasks executions are interpreted as the users' cost functions, which are specified as the scheduling objectives and are minimized during the game. The main drawback of the online scheduling approaches may be the high computational complexity of the schedulers. In many cases the games are provided on the different grid levels and the design of an effective synchronization mechanism is a challenging task. A game-theoretical support to the users' decisions and actions will be discussed in Chapter 6.

Artificial Neural Networks (ANNs) are usually implemented as schedulers in grid computing. An illustrating example can be the grid scheduler based on the Fuzzy Neural Networks presented in [166]. The authors used the fuzzy logic module for monitoring the status of machine loads in grid system. The parameters of fuzzy membership functions in this model are tuned by using the ANN trained by back-propagation algorithm.

In [131] the ANN mechanism is used for supporting the users' decisions. The authors defined a decision model which is composed of three main components: (a)

online module for the prediction of the users' actions; (b) off-line module for the analysis of statistical data acquired during user's work; and (c) 'users activity' module defined for the detection of trends and changes in users' activities. The users' decisions mechanisms are supported by the feed-forward neural networks trained by the back-propagation method. The authors additionally proposed the offline model, where another neural network is applied for the detection of normal/abnormal users activities, by analyzing the statistical data accumulated during the users' actions.

The above mentioned model is a promising solution for simulation and simple analysis of the users' decisions. This model can be considered as an alternative to game-based methodology in online scheduling. However the high complexity of this model can be a main drawback for its successful application in real-life grid scenarios.

5.3 Security as Scheduling Criterion in Computational Grids

A general security-aware grid model is based on the hierarchical multi-level architecture presented in Chapter 1 (see Sec. 1.2.2). However, the role of the meta-scheduler is different when security is considered as additional criterion in the scheduling process. The meta-scheduler must analyze the security requirements for the execution of tasks and requests of the CG users for trustful resources available within the system. The system brokers analyze "reputation" indexes of the machines received from the resource managers and send proposals to the scheduler. Moreover, the brokers also control the resource allocation and communication between CG users and resource owners.

Fig. 5.1 depicts the 3-level architecture of the security-aware grid cluster.

The trust level and security demand parameters are generated by aggregation of several scheduling and system attributes. Those parameters depend heavily on the security policy, accumulated resource or grid cluster "reputation", self-defense capability, attack history, special users' requirements, and peer authentication. Fig. 5.2 presents the major behavior and intrinsic security attributes needed for the specification of trust levels of the grid clusters and security demand of the grid applications (see also [137]).

Song et al. in [136] have developed a fuzzy-logic trust model, in which the aforementioned attributes are aggregated into single scalar parameters. The task security demand in this model is supplied by the user's programs as request for authentication, data encryption, access control, etc. The trust level parameters of the resource clusters are aggregated through a **two-level** hierarchic fuzzy-logic based trust procedure in the following way:

- At the lower *intra-site* level there are applied two fuzzy inference systems for the evaluation of the self-defence capabilities and trust indexes of the resources; each grid cluster reports its assessed self-defense capability to all other clusters;
- At the higher *inter-site* level there are collected the inputs from all resource clusters and the trust level vector is defined through another fuzzy inference process (see [136]).

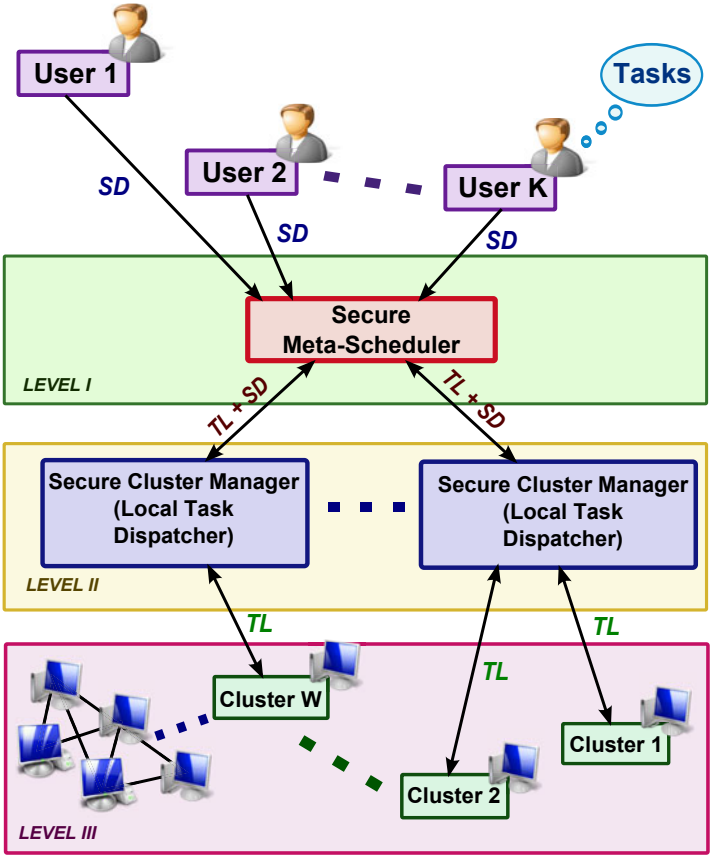


Fig. 5.1 The model of secure grid cluster

This fuzzy trust model was used in this work for the specification of new characteristics of tasks and resources in the grid system, namely security demand and trust level vectors. The *security demand vector* , denoted by $SD = [sd_1, \ldots, sd_n]$ sd_j, is defined as a vector of the security demand parameters sd_j, $(j \in N)$, for all tasks in the batch. The *trust level vector* , denoted by $TL = [tl_1, \ldots, tl_m]$, is defined as a vector of trust level parameters tl_i for all resources in the system. The trust level parameters specify how much a grid user can trust the resource manager. The manager maintains machine i status and monitors the execution of the tasks assigned to this machine. The values of the sd_j and tl_i parameters are real fractions within the range [0,1] with 0 representing the lowest and 1 the highest security requirements for a task execution and the most risky and fully trusted machine, respectively. A task can be successfully completed at a resource when a *security assurance condition* is satisfied. That is to say that $sd_j \leq tl_i$ for a given (j,i) task-machine pair.

Let us denote Pr_f to be a *Machine Failure Probability* matrix , the elements of which, are interpreted as the probabilities of failures of the machines during the tasks

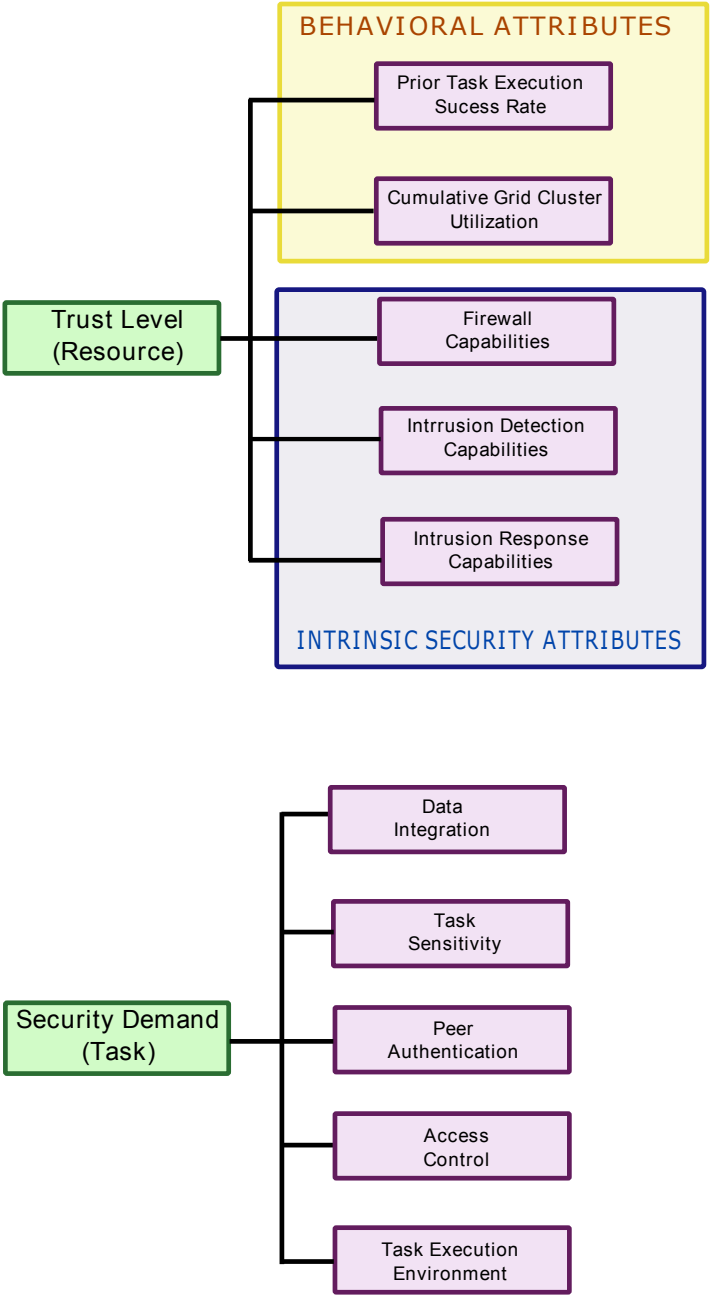


Fig. 5.2 The major attributes affecting the trust level and security demand in grid systems

executions due to the high security restrictions. These probabilities are denoted by $P_f[j][i]$ and are calculated by using the negative exponential distribution function, that is to say:

$$Pr_f[j][i] = \begin{cases} 0 & , sd_j \leq tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & , sd_j > tl_i \end{cases} \tag{5.2}$$

where α is interpreted as a failure coefficient and is a global parameter of the model.

The process of matching sd_j with tl_i is similar to that of a real-life scenario where users of some portals, such as Yahoo!, are required to specify the security level of the login session.

5.3.1 Scheduling Scenarios and Objectives

The grid cluster or the grid resource may be not accessible to the global meta-scheduler when being infected with intrusions or by malicious attacks. The scheduler has two options of initializing his work: (a) to analyze the *Machine Failure Probability* matrix in order to minimize the failure probabilities for task-machine pairs; or (b) to perform an "ordinary" scheduling without any preliminary analysis of the security conditions, abort the task scheduling in the case of machine failure, and reschedule this task at another resource. The scheduler's strategies give rise to two modes of processing (and modelling in particular) the grid schedules, namely *secure* and *risky* modes.

Secure Mode

In this scenario all of the security and resource reliability conditions are verified for all task-machine pairs. The main goal of the meta-scheduler is to design an optimal schedule for which, beyond the makespan and flowtime, the probabilities of failures of the machines during the tasks executions will be minimal. It is assumed that additional "cost" of the verification of security assurance condition for a given task-machine pair: (a) may delay the predicted execution time of the task on the machine, and (b) is proportional to the probability of failure of the machine during the task execution. This "cost" is defined as the product of $Pr_f[j][i]$ and $ETC[j][i]$. In this case the completion time of the machine i is denoted by $completion^s[i]$[1] and can be calculated as follows:

$$completion^s[i] = ready_i + \sum_{\{j \in Tasks(i)\}} (1 + Pr_f[j][i])ETC[j][i]) \tag{5.3}$$

where $Tasks(i)$ denotes the set of tasks assigned to the machine i in a given batch.

In this mode the main scheduling objectives, namely makespan and flowtime, can be expressed as follows:

[1] The general concept of the completion time of machine was explained in Sec. 2.2.2 in Chapter 2.

$$C_{max}(sec) = \max_{i \in M} completion^s[i]. \tag{5.4}$$

$$F(sec) = \sum_{i \in M} F^s[i] \tag{5.5}$$

where

$$F^s[i] = ready_i + \sum_{j \in Sorted[i]} (1 + Pr_f[j][i])ETC[j][i]) \tag{5.6}$$

and $Sorted[i]$ denotes the set tasks assigned to the machine i sorted in ascending order by the corresponding ETC values.

Risky Mode

In this scenario all secure and failing conditions are ignored. The scheduling process is realized as a two-step procedure. First, the scheduling is performed just by analyzing the ETC matrix. If failures of machines are observed, then the unfinished tasks are temporarily moved into the backlog set. This set is defined as a 'batch supplement' and the tasks form this set are re-scheduled in the way as in the secure mode. The total completion time of machine $i(i \in M)$ in this case can be defined as follows:

$$completion^r[i] = completion[i] + completion^s_{res}[i] \tag{5.7}$$

where $completion[i]$ is calculated by using the Eq. (2.13)(see Chapter 2, Sec. 2.2.2), for tasks primarily assigned to the machine i , and $completion^s_{res}[i]$ is the completion time of machine i calculated by using the Eq.(5.3) for rescheduled tasks, i.e. the tasks re-assigned to the machine i from the other resources.

The formulas for makespan and flowtime in this mode are defined in the following way:

$$C_{max}(risk) = \max_{i \in M} completion^r[i]. \tag{5.8}$$

$$F(risk) = \sum_{i \in M} F^r[i] \tag{5.9}$$

where

$$F^r[i] = ready_i + \sum_{j \in Sorted[i]} ETC[j][i] + \sum_{j \in Sorted_{res}[i]} (1 + Pr_f[j][i])ETC[j][i]) \tag{5.10}$$

and and $Sorted_{res}[i]$ denotes the set of *rescheduled* tasks assigned to the machine i sorted in ascending order by the corresponding ETC values

Assuming the hierarchical optimization mode (see Eq. (5.1), parameter *hier*) with the makespan as the primarily scheduling criterion, the flowtime should be minimized in both secure and risky scenarios subject to the the following constraints:

- in the secure mode

$$F^s[i] \leq C_{max}(sec)\ \forall i \in M; \tag{5.11}$$

- in the risky mode

$$F^r[i] \leq C_{max}(risk)\ \forall i \in M. \tag{5.12}$$

Although the probabilities of machines' failures are expected to be higher in the risky than in the secure mode, there is certainly no guarantee of the successful execution of all tasks in the security scenario. It can be observed that if the *security assurance condition* is satisfied for each task-machine pair (i.e. $sd_j \leq tl_i$ for $i \in M, j \in N$), the completion times of machines in both *secure* and *risky* modes are identical with the completion times defined for standard independent scheduling problem (see Chapter 2, Eq.(2.13)), where it is assumed that each task *must* be successfully executed on each machine and no security requirements are analyzed[2].

5.4 Artificial Neural Network Module

The implementation of Artificial Neural Network (ANN) module requires preliminary classification of tasks and machines available in the system. This classification is based on the values of the workload (WL), computing capacity (CC), trust level (TL) and security demand (SD) vectors. Machines are categorized into the R_r types according to their *processing power* features, namely *slowest, slower,*$\cdots$*,medium,*$\cdots$*,fastest* classes; and into R_s types according their *trust level* features, namely *secure, less_secure,*$\cdots$*,medium,*$\cdots$*,fully_risky* classes. This initial classification leads to the overall categorization of the resources into $R = R_r \cdot R_s$ classes, namely *slowest-secure,*$\cdots$*,medium-secure,*$\cdots$*,fastest-fully_risky* types, in order to characterize the grid machine under the processing power and trust criteria.

Similar classification can be provided for the submitted tasks under workload and security demand features. The tasks are categorized into $T = T_w \cdot T_{sd}$ types, where T_w is number of *workload* classes and T_{sd} is number of *security demand* classes. R machine classes and T task classes generate $R + T$ possible inputs for neural network.

Formally, the ANN input data can be expressed by the following pair of vectors: $\{TASKS\_MX; MACHINES\_MX\}$, where:

- $TASKS\_MX[\widehat{t}] = T_t$ for tasks classification, where $\widehat{t}$ denotes a task class, ($t = 1, \ldots, T$), and T_t denotes the fraction of tasks in the class $\widehat{t}$. That is to say:

$$T_t = \frac{\widehat{t_t}}{n}, \tag{5.13}$$

[2] The component $completion^s_{res}[i]$ in Eq. (5.7) is not calculated while all tasks are successfully executed on the grid machines.

where $\widehat{t_t}$ is the number of tasks in the class $\widehat{t}$ and

$$\sum_{t=1}^{T} T_t = 1 \tag{5.14}$$

- $MACHINES\_MX[\widehat{r}] = R_r$ for resources classification, where $\widehat{r}$ denotes a machine class ($\widehat{r} = 1, \ldots, R$), and proportion of machines in the class $\widehat{r}$ is denoted by R_r. That is to say:

$$R_r = \frac{\widehat{r_r}}{m}, \tag{5.15}$$

where $\widehat{r_r}$ is the number of machines in the class $\widehat{r}$ and

$$\sum_{r=1}^{R} R_r = 1 \tag{5.16}$$

ANN module monitors the machine failures and the successful execution of tasks on machines. The information about the failures of the grid resources is needed for the classification of the results generated by the neural network. For each class $\widehat{r}$ of machines, there is selected a unique *major class* $t_{maj}(\widehat{r})$ of successfully executed tasks. This class is specified based on the number of completed tasks on a given machine without any failures and re-scheduling procedures. The results generated by the neural network are defined as an *output matrix* OUT_MX of the size $T \cdot R$ with just R non-zero (positive) elements (one major class of tasks for each host is indicated in such a way), where:

$$OUT\_MX[\widehat{r}][t_{maj}(\widehat{r})] = r_{(t_{maj}(\widehat{r}))} \tag{5.17}$$

and $r_{(t_{maj}(\widehat{r}))}$ is the proportion of the tasks from the major class $t_{maj}(\widehat{r})$ submitted to the machines of the class $\widehat{r}$. The main concept of the network is presented in Fig. 5.3.

The network is trained by the *back propagation* algorithm [61]. The results generated by the ANN module are used for the specification of the grid schedules, which may be accepted as the partial (or optimal) solutions for the problem, or may be passed on to the heuristic or meta-heuristic schedulers as the initial solutions. The procedure of the generation of the schedules based on the ANN "suggestions" can be defined as follows. First, a 'major class' $t_{maj}(\widehat{r})$ membership is verified for all tasks in the batch. For each task from the 'major class' the class of the fastest and most trustful machines is selected. Thereafter, this task is assigned to the machine from the selected class, with the minimal completion time. The tasks from the other classes than the 'major' one are assigned to the machines with the shortest completion times without analyzing the network results. The *Minimum Completion Time (MCT)* method is used for all those assignments. The general framework of MCT procedure is presented in Alg. 2.

Both input and output matrices defined for ANN are totally independent of numbers of hosts and tasks in the system. Therefore, the ANN module can be trained even on a small batch of task and small cluster of machines and then the generated results may be used in more complex scenarios.

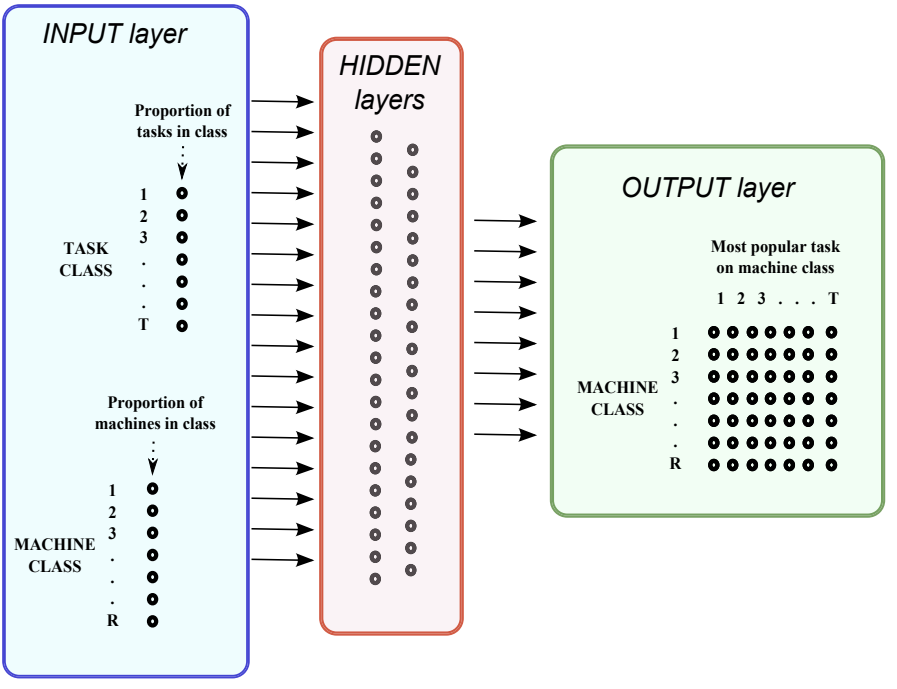


Fig. 5.3 Neural network architecture

Algorithm 2. *MCT* algorithm template

1: Calculate the *ready_times* of the machines ;
2: **for all** Tasks in a given batch **do**
3: Calculate the completion times of the machines for the tasks;
4: Find the machine that gives minimum completion time, m_{best};
5: Assign task to m_{best} machine;
6: Update the machine completion time;
7: **end for**
8: Return the resulting schedule

5.5 Empirical Evaluation of the Genetic Metaheuristics for Security-Aware Scheduling

The specification of just one major class of tasks for the network results may be of course the main reason of relatively low efficiency of the ANN module in some realistic approaches. The intelligent modification of this neural network model into the low-cost (in the sense of the execution time) multi-class version remains still challenging research issue. However, the monitoring of the system, the provisioning of the resources and scheduling according to various security requirements, should

be even partially automated. Otherwise, the information about the reliability of the resources, all negotiations of the particular security conditions, or simply the information about the failures of the nodes, must be proceed by the system administrators and grid users. It usually delays the realization of the schedules because of the sheer scale of the system and different, and often conflicting, goals and interests of the users working at the various system levels. The ANN model presented in this chapter may serve as a prototype solution for both automatic monitoring and detection of the system failures and intelligent supporting of the users decisions, with a tangible benefit of reducing an overall computing overhead in high-performance computing systems (not just grids).

The aim of the empirical study presented in this section is the verification of the efficiency of the neural network support for single- and multi-population genetic-based schedulers in the reduction of the number of failures of the resources caused by too restrictive security conditions, and in the minimization of two conventional scheduling objectives, namely makespan and flowtime. The experiments were conducted in two main steps. First, six variants of single-population GA-based schedulers with different crossover, mutation and replacement operators were evaluated in both scheduling modes. The goal of this analysis is to compose an optimal combination of genetic operators for multi-population hierarchical, island and hybrid genetic schedulers. Thereafter, four genetic meta-heuristics were evaluated in risky and secure modes, namely the best single-population GA in the first part of the analysis, and *HGS-Sched*, *Island GA* and hybridization of GA with Tabu Search (*GA+TS*).

5.5.1 Security Aware Sim-G-Batch Grid Simulator

The ANN module was integrated with the *Sim-G-Batch* simulator for modelling and monitoring the grid system behavior under the specified security conditions. The module works in the "background" of the main system and supports the resolution methods implemented in *Scheduler* class. The sub-optimal schedules generated based on the neural networks "suggestions", are passed on to the initial populations of the genetic schedulers. The main concept of the security-aware version of *Sim-G-Batch* simulator with the ANN module is presented in Fig. 5.4.

In the case of security scheduling, the list of typical input parameters for *Sim-G-Batch* (see Chapter 2, Sec. 2.3.1) is extended by the trust level vector TL and the security demand vector SD. Table 5.1 presents the values of key parameters in four grid size scenarios, namely *Small, Medium, Large* and *Very Large* in static and dynamic modes. Most of those parameters (excluding the numbers of tasks and machines) were tuned in empirical analysis presented in [136], [137], [92] and the recent publications of the author of this book [18], [87], [89].

For activating the ANN module, the tasks and machines are divided into 18 classes: 9 categories for processing power and trust level criteria (machines), and 9 categories for workload and security demand criteria (tasks). The ANN is a feed-forward network with two hidden layers, the weight coefficients are in the range of $[-0.2; 0.2]$ and the learning rate is 0.01. The training set for ANN contains the

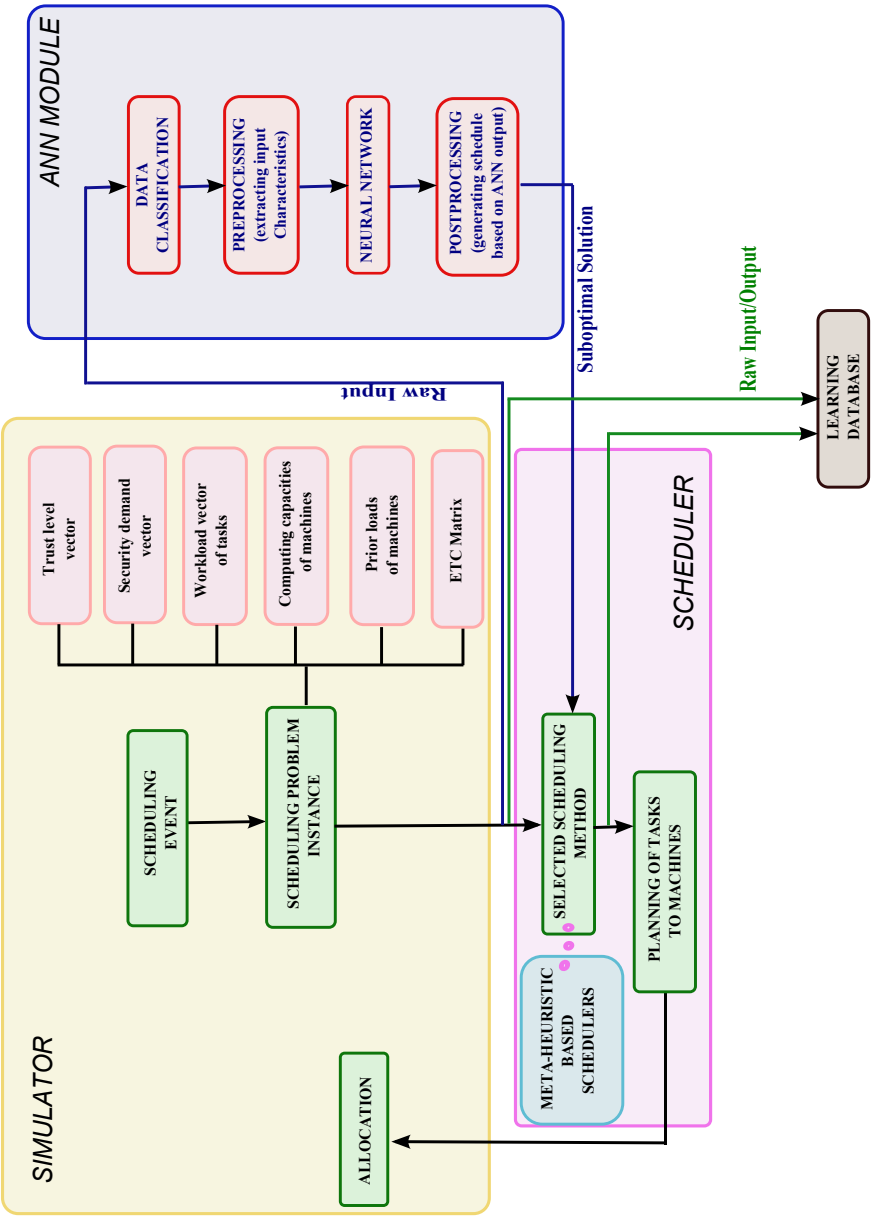


Fig. 5.4 General Flowchart of the secure *Sim-G-Batch* Simulator Supported by Neural Network Linked to Scheduling

Table 5.1 Values of key parameters of the secure *Sim-G-Batch* in static and dynamic modes

	Small	**Medium**	**Large**	**Very Large**
		Static case		
Nb. of hosts	32	64	128	256
Resource cap. (in MHz CPU)		$N(5000, 875)$		
Total nb. of tasks	512	1024	2048	4096
Workload of tasks		$N(250000000, 43750000)$		
Security demands sd_j		$U[0.6; 0.9]$		
Trust levels tl_i		$U[0.3; 1]$		
Failure coefficient α		3		
		Dynamic case		
Init. hosts	32	64	128	256
Max. hosts	37	70	135	264
Min. hosts	27	58	121	248
Resource cap. (in MHz CPU)		$N(5000, 875)$		
Add host	$N(625000, 93750)$	$N(562500, 84375)$	$N(500000, 75000)$	$N(437500, 65625)$
Delete host		$N(625000, 93750)$		
Init. tasks	384	768	1536	3072
Total tasks	512	1024	2048	4096
Inter arrival	$E(7812.5)$	$E(3906.25)$	$E(1953.125)$	$E(976.5625)$
Workload		$N(250000000, 43750000)$		
Security demands sd_j		$U[0.6; 0.9]$		
Trust levels tl_i		$U[0.3; 1]$		
Failure coefficient α		3		

characteristics of the tasks and machines and the task-machine matching results collected after the 500 runs of the simulator with inactive Neural Network module.

5.5.2 Performance Measures

The performances of all schedulers in experiments were evaluated under the following three metrics:

- *Makespan* – a primarily scheduling criterion, which is expressed in Eq. (5.8) in risky scenario, and in Eq. (5.4) in the secure mode,
- *Mean_Flowtime* – flowtime scheduling criterion, which is defined in Eq. (5.8) for the risky mode, and in Eq. (5.4) for the secure mode; and
- *FailureRate* $Fail_r$ parameter defined as follows:

$$Fail_r = \frac{n_{failed}}{n} \cdot 100\% \tag{5.18}$$

where n_{failed} is the number of unfinished tasks, which must be rescheduled[3].

Both *Makespan* and *Mean_Flowtime* measures are expressed in arbitrary time units specified for the scheduling.

5.5.3 Tuning the Genetic Engine for Multi-Population Batch Schedulers

In the first part of empirical study six single-population risk-resilient GA schedulers have been compared in order to define an effective genetic engine for multi-population meta-heuristics. The configuration of the genetic parameters for those six schedulers are presented in Table 5.2.

Table 5.2 Configuration of six single-population GA-based grid schedulers

Scheduler	Replacement method	Scheduling scenario
GA-SS-R	Steady State	Risky
GA-SS-S	Steady State	Secure
GA-SS-ANN	Steady State	Secure supported by ANN
GA-ST-R	Struggle	Risky
GA-ST-S	Struggle	Secure
GA-ST-ANN	Struggle	Secure supported by ANN

The aforementioned methodologies differ in the implementation of the replacement mechanism. The *Steady State* replacement method is used in *GA-SS-xxx*

[3] According the notation introduced in Chapter 2, n stands for the number of tasks in a given batch.

algorithms and *Struggle* procedure – in *GA-ST-xxx*. The ANN module is active just in *GA-SS-ANN* and *GA-ST-ANN* algorithms for generating a part of an initial population. All of the remaining procedures in these algorithms are identical with the schedulers working in the secure scenario. Based on the results of the tuning process of genetic-based meta-heuristics in conventional grid scheduling presented in Chapter 4, Sec. 4.4.1, the remaining genetic operations in the schedulers are configured as follows: (i) *Linear Ranking* as selection scheme, (ii) *Cycle Crossover (CX)* operator and (iii) *Move* mutation method [107].

The generic frameworks of all considered schedulers are the same as in Alg. 1 defined in Chapter 3. The key parameters of HGS-Sched model for generating all types of genetic-based schedulers are presented in Table 5.3.

Table 5.3 GA Steady State and GA Struggle settings in static and dynamic cases

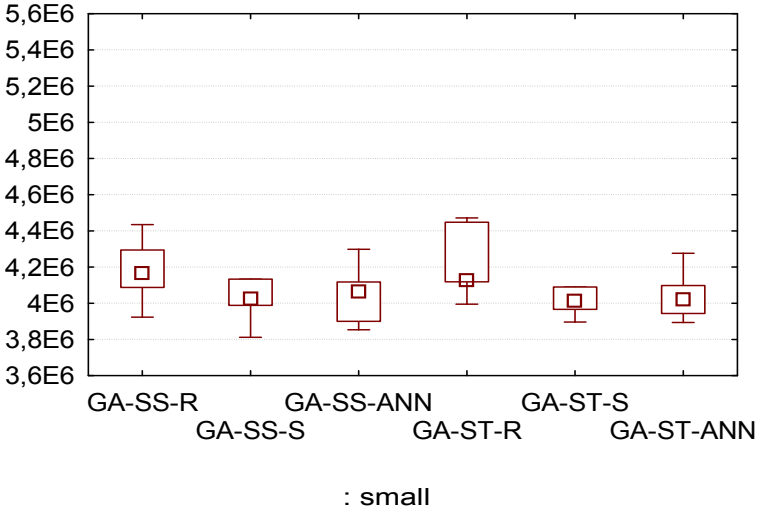
Parameter	**Value**
degree of branches (t)	0
period_of_metaepoch (α)	$(5*n)/10$
nb_of_metaepochs	10
population size (pop_size)	60
intermediate pop.	48
cross probab.	0.9
mutation probab.	0.15
$max\_time\_to\_spend$	200 sec. $(static)$ 400 sec. $(dynamic)$

The *HGS-Sched* has in this case just one core branch. Each experiment was repeated 30 times under the same configuration of operators and parameters.

5.5.3.1 Results

The results of the minimization of the *Makespan* in both *risky* and *secure* modes and all grid scenarios, are presented as the box-plots in Fig. 5.5–5.8.

The best results in the *Makespan* optimization have been achieved by both *GA-XX-ANN* schedulers. The efficiency of the ANN support can be observed especially



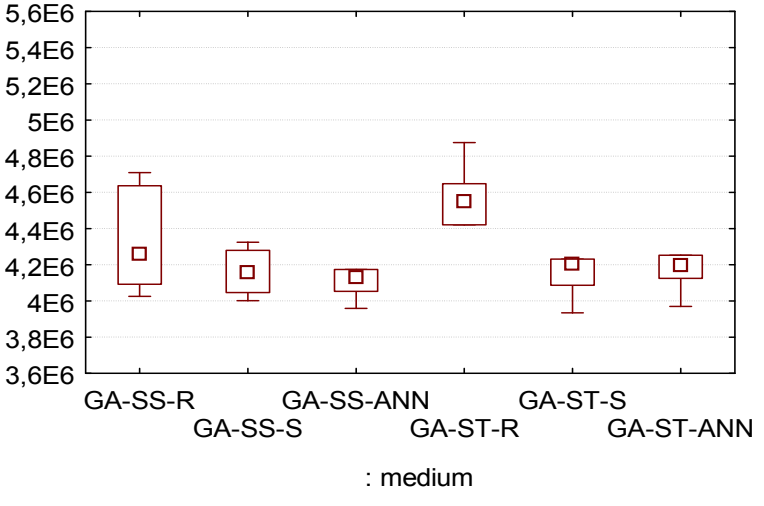


Fig. 5.5 The box-plot of the results for *Makespan* in static scheduling: Small and Medium grids

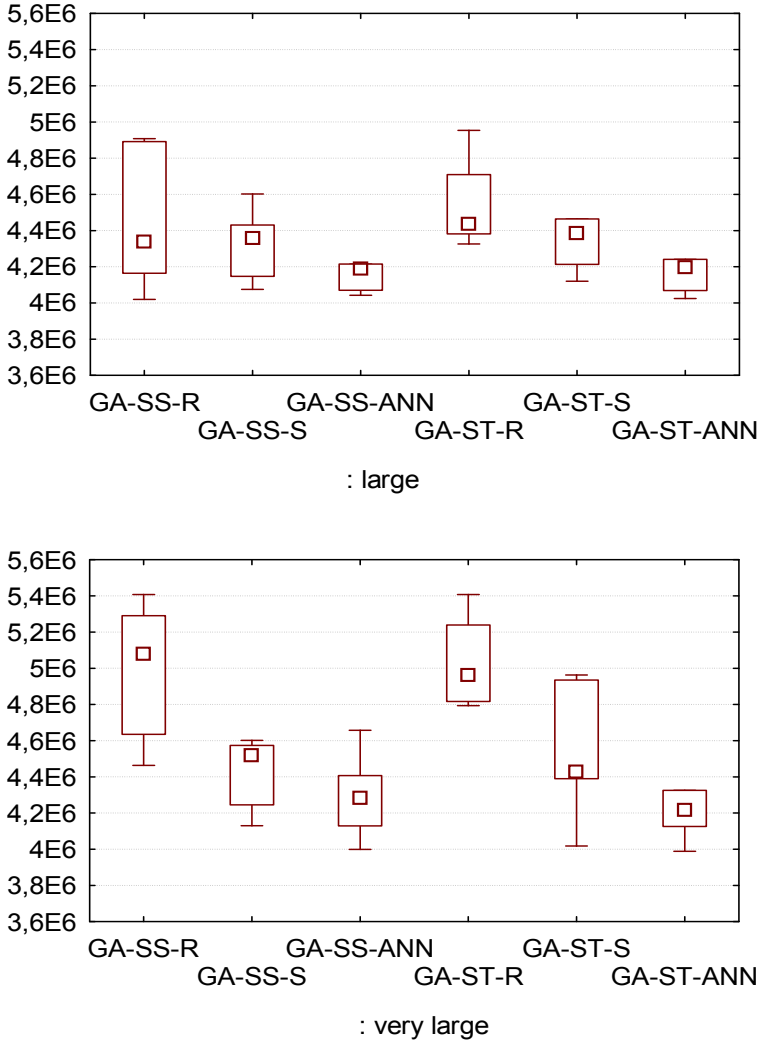


Fig. 5.6 The box-plot of the results for *Makespan* in static scheduling: Large and Very Large grids

in the 'Large' and 'Very Large' grid scenarios. The worst in the *Makespan* reduction were the schedulers working in the risky mode. While in the 'Small' grid case the differences in the averaged *Makespan* values are not so big, in all other scenarios *GA-SS-R* and *GA-ST-R* meta-heuristics significantly lag behind the secure schedulers. It can be also observed that in all instances the distribution of the results are asymmetric and the medians are very close to the first or the third quantiles.

The box-plots of the *Mean_Folwtime* results are presented in Fig. 5.9–5.12.

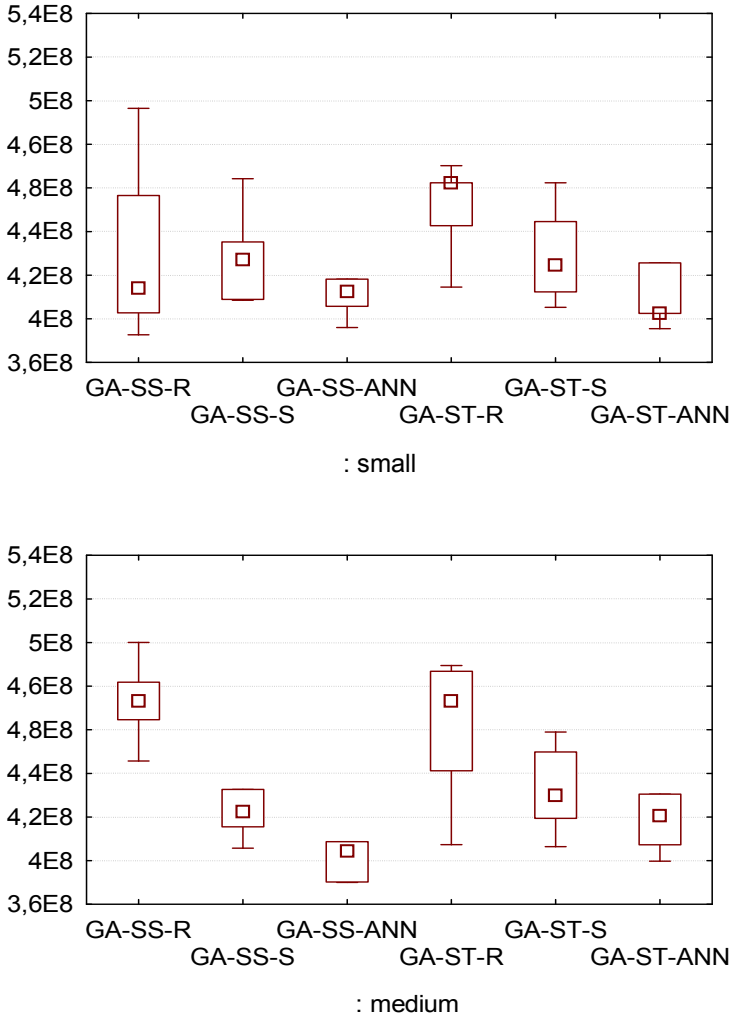
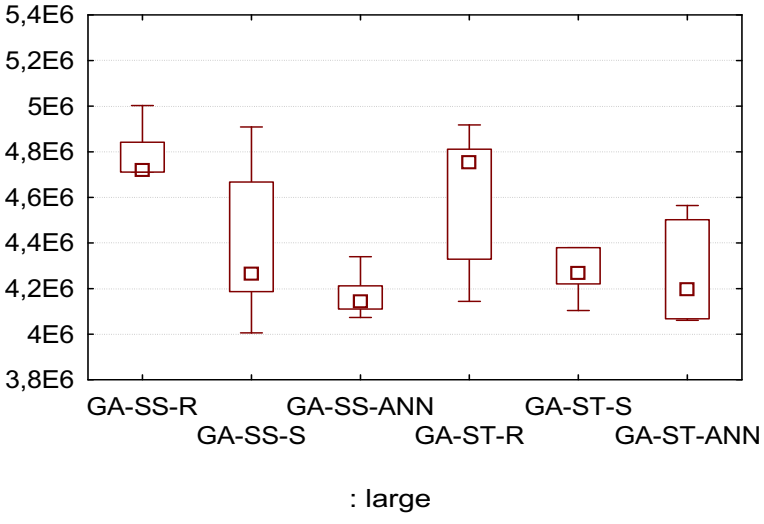


Fig. 5.7 The box-plot of the results for *Makespan* in dynamic scheduling: Small and Medium grids

In the case of the minimization of the flowtime, both *GA-XX-ANN* meta-heuristics outperform again the rest of the methods in both static and dynamic modes, however the differences in the results are not so significant as it was in the *Makespan* case. Additionally, it can be noted that as the instance size is doubled, the *Mean_Flowtime* values increase considerably for all applied schedulers, while the *Makespan* is almost at the same level. Another observation is that all schedulers are rather 'symmetric' in the sense of the distribution of the results and the differences between the first and the third quantiles are rather small. The best relative effectiveness of the ANN support may be observed in 'Very Large' grids in static and dynamic cases.



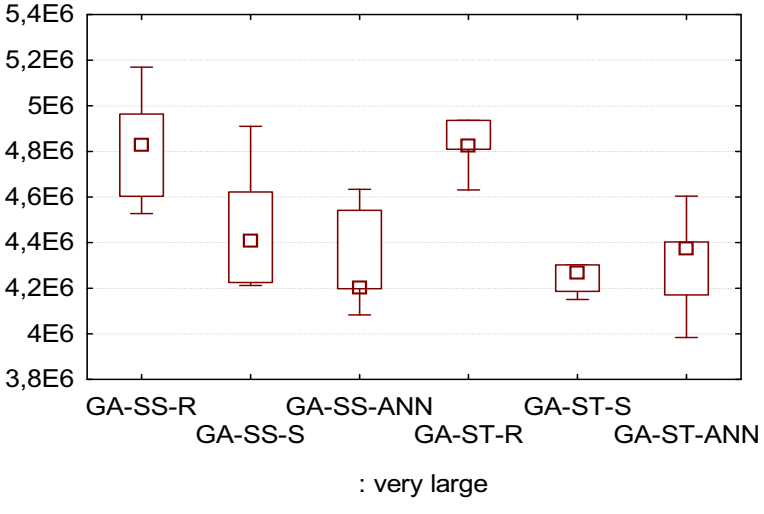


Fig. 5.8 The box-plot of the results for *Makespan* in dynamic scheduling: Large and Very Large grids

Similarly to other experiments presented in this book, the maximal number of generations in GA was defined as a stopping criterion for all schedulers. However, the solutions generated by ANN may not be improved by the GA meta-heuristics, and the search process can be stopped because of the stagnation in the improvement of the solutions' quality. Table 5.4 presents the averaged (in 30 runs of the simulator) minimal numbers of genetic epochs (generations) needed for generating the best solutions by all considered genetic schedulers. A relative effectiveness of each

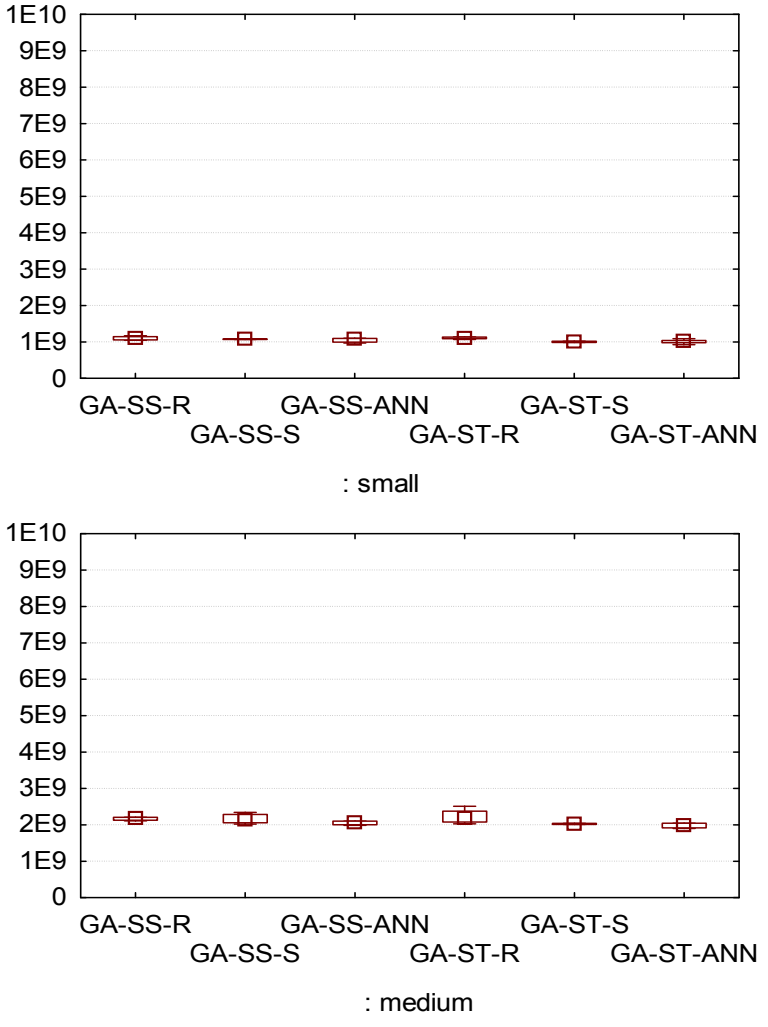


Fig. 5.9 The box-plot of the results for *Mean_Folwtime* in static scheduling: Small and Medium grids

scheduler is expressed as the ratio of the minimal number of genetic epochs necessary for finding the optimal solutions, and the stopping criterion, which is $5 \cdot n$, where n denotes the number of tasks in the system. These parameters are displayed in parentheses in Table 5.4.

It can be noted than ANN module in most of the instances reduced the time necessary for finding the best solutions approximately by 30–40 %, and successfully speeded up the search process in both secure and risky scenarios. The effectiveness of the ANN support is confirmed by the lowest failure rates achieved by the *GA-XX-ANN* schedulers. The results for all six schedulers are presented in Table 5.5.

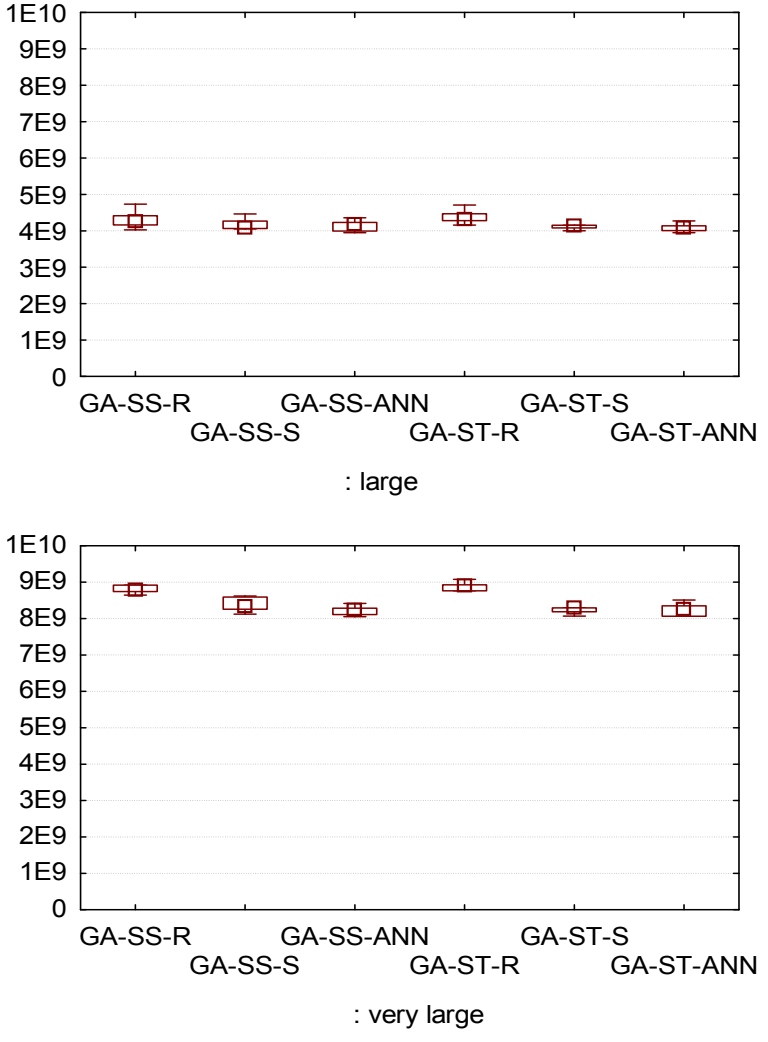


Fig. 5.10 The box-plot of the results for *Mean_Folwtime* in static scheduling: Large and Very Large grids

In all instances but one – the 'Small' grid in static scenario – the schedulers with the active ANN module outperform the other methods. The ANN support allow to reduce the machine failures by $1\% - -6\%$ compared to the 'conventional' schedulers.

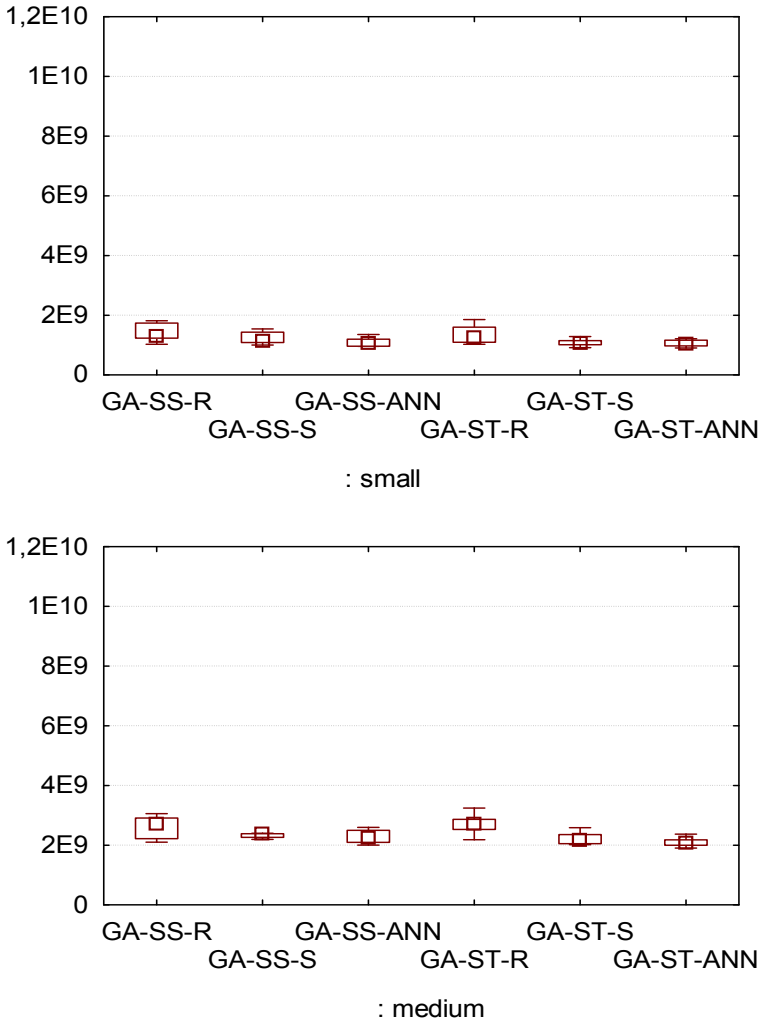


Fig. 5.11 The box-plot of the results for *Mean_Folwtime* in dynamic scheduling: Small and Medium grids

5.5.3.2 Summary

Based on the results of all experiments provided for single-population GA schedulers, the most effective in the optimization of both scheduling objective functions is *GA-SS-ANN*. This algorithm works in the secure mode with the ANN support and *steady state* replacement mechanism in the main genetic engine. This algorithm achieved the lowest failure rates in half of the instances, and most of them in the dynamic grid, which makes it the best candidate methodology for the secure

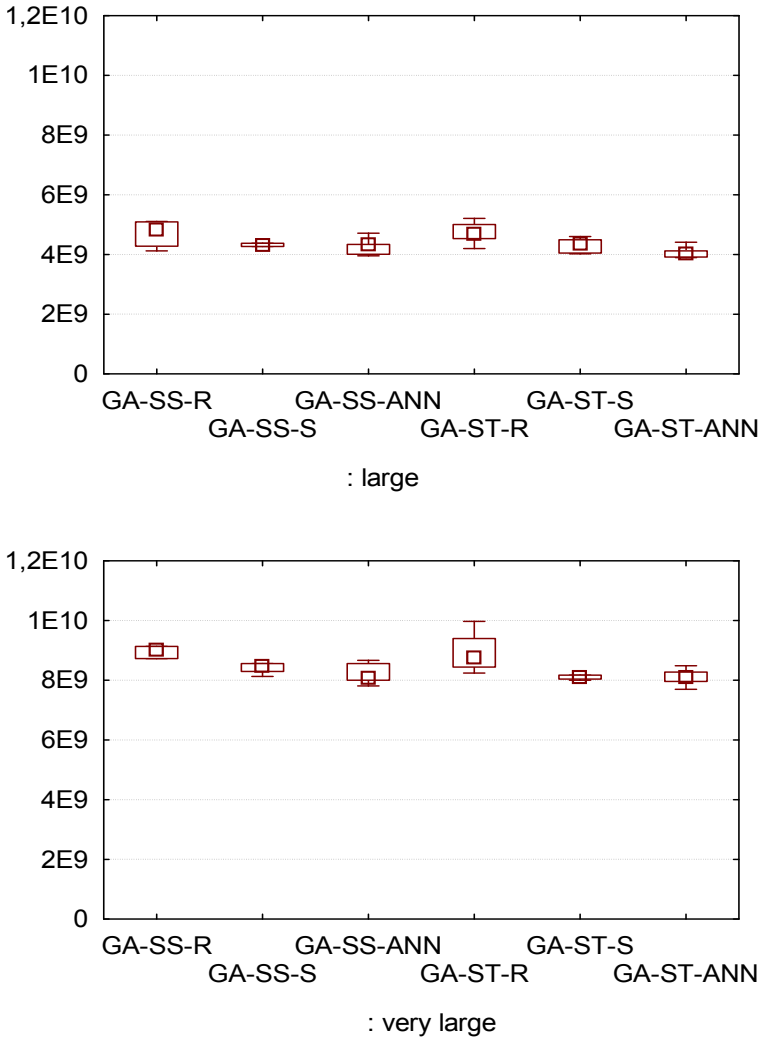


Fig. 5.12 The box-plot of the results for *Mean_Folwtime* in dynamic scheduling: Large and Very Large grids

scheduling in the realistic scenario. It is also the best in the minimization of the *Makespan* and flowtime in most of the instances of the scheduling problem.

It can be observed that generally it is more resilient for the grid schedulers to 'pay' a priori some additional scheduling 'cost' due to verification of the security conditions, than taking a risk on allocating the unreliable resources. As a result, the failure rates in the risky mode are much higher than in the secure case, especially in the dynamic grid where the frequency of the machine failures are 3–4 times greater than in the secure scenario. This is the main reason of lower effectiveness of

Table 5.4 The averaged minimal numbers of genetic epochs necessary for generating the best solutions by six considered GA-based schedulers

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-R	2302 (89.92%)	4722 (92.22%)	10008 (97.73%)	19226 (93.87%)
GA-SS-S	2031 (79.33%)	3620 (70.70%)	8345 (81.49%)	19740 (96.38%)
GA-SS-ANN	1722 (67.26%)	**2733** (53.37%)	**7992 (78.04%)**	17739 (86.61%)
GA-ST-R	1923 (75.11%)	4213 (82.28%)	10013 (97.78%)	20054 (97.91%)
GA-ST-S	1987 (77.61%)	4005 (78.22%)	8022 (78.33%)	18654 (91.08%)
GA-ST-ANN	**1592 (62.18%)**	3872 (75.62%)	8591 (83.89%)	**16940 (82.71%)**
		Dynamic Instances		
GA-SS-R	2090 (83.6%)	5099 (99.78%)	10100 (98.63%)	20145 (98.75%)
GA-SS-S	1831 (71.52%)	3925 (76.96%)	9036 (90.36%)	19002 (92.78%)
GA-SS-ANN	**1522 (60.60%)**	3021 (59.23%)	8010 (78.52%)	**17830** (87.06%)
GA-ST-R	2175 (85.21%)	4923 (96.15%)	10057 (98.59%)	19353 (94.86%)
GA-ST-S	1703 (68.52%)	**2954 (57.92%)**	8238 (80.70%)	17993 (88.63%)
GA-ST-ANN	1611 (61.44%)	3401 (66.68%)	**6035** (60.78%)	17910 (87.83%)

the schedulers in the optimization of the main grid objective functions in the risky mode. The ANN support in the security scheduling allow to reduce significantly the *Makespan* and *Mean_Folwtime* values and to keep the failure rates of the machines at the sufficiently low levels.

Table 5.5 Average values of failure rate $Fail_r$ parameter for six GA-based schedulers [$\pm s.d.$] ($s.d.$ = standard deviation)

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-R	4.832% [± 0.97]	7.201% [± 0.78]	11.824% [± 1.26]	31.721% [± 3.28]
GA-SS-S	4.008% [± 1.15]	4.135% [± 1.27]	10.698% [± 3.26]	11.635% [± 3.13]
GA-SS-ANN	3.993% [± 0.98]	**4.089%** [± 1.56]	8.436% [± 1.67]	**8.736%** [± 2.09]
GA-ST-R	4.697% [± 1.71]	17.516% [± 3.39]	14.013% [± 4.08]	35.643% [± 6.73]
GA-ST-S	**3.897%** [± 0.96]	5.540% [± 1.89]	10.945% [± 1.63]	10.402% [± 3.42]
GA-ST-ANN	3.967% [± 0.79]	6.430% [± 0.63]	**6.11%** [± 1.28]	9.196% [± 2.77]
		Dynamic Instances		
GA-SS-R	12.126% [± 1.80]	25.306% [± 2.79]	31.342% [± 3.44]	25.794% [± 2.48]
GA-SS-S	6.104% [± 1.69]	6.916% [± 2.40]	9.507% [± 1.84]	8.943% [± 2.07]
GA-SS-ANN	4.880% [± 0.98]	**6.097%** [± 1.62]	**7.456%** [± 1.32]	**7.026%** [± 2.11]
GA-ST-R	26.797% [± 5.25]	22.96% [± 4.19]	29.227% [± 4.95]	29.380% [± 5.30]
GA-ST-S	9.218% [± 2.84]	7.623% [± 2.02]	8.084 % [± 2.49]	9.744% [± 2.69]
GA-ST-ANN	**4.093%** [± 0.97]	6.991% [± 1.44]	7.681% [± 1.33]	7.894% [± 2.41]

5.5.4 Evaluation of Multi-Population and Hybrid Genetic Metaheuristics

The effectiveness of the multi-population meta-heuristics and hybrid genetic schedulers depend on the efficiency of their single-population genetic engines. $GA-SS-ANN$ algorithm, as the best single-population GA in the first part of the empirical analysis, was selected to serve as the main genetic mechanism in HGS-Sched, Island GA and $GA+TS$ hybrid algorithms applied to the secure grid scheduling. The following four meta-heuristics were considered in this part of analysis:

- *GA-SS-ANN* - with the settings defined in Table 5.3;
- *Sec-HGS-Sched* - with $GA-SS-ANN$ engine and various population sizes and mutation rates in the branches of different degrees;
- *Sec-IGA* - Island Genetic Algorithm with $GA-SS-ANN$ as the basic mechanism in all sub-populations;
- *Sec-(GA+TS)* - hybrid scheduler with $GA-SS-ANN$ as the control strategy and Tabu Search (TS).

The general characteristics of hierarchical, island and hybrid schedulers are presented in Sec. 4.4.2.1 in Chapter 4. The settings for all considered meta-heuristics are the same as the values of global parameters for *IGA*, *HGS-Sched* and *GA+TS* presented defined in Tables 4.16, 4.17 and 4.18. It means that *Sec-HGS-Sched* is composed of one branch of degree 0 and branches of degrees 1.

5.5.4.1 Results

The results of the comparative analysis of the minimization of *Makespan*, $Mean\_Folwtime$ and the failure rates in static and dynamic instances are presented in Tables 5.6, 5.7 and 5.8. The results were averaged over the 30 runs of the simulator for the same configuration of schedulers and all parameters.

The results show the high effectiveness of the hierarchic scheduler in the security-aware scheduling. *Sec-HGS-Sched* achieved the best results in 80 % of the instances for all scheduling metrics. It is the best in the reduction of the failing rates in 7 cases, which makes this model a solid base for the development of the real-life scheduling strategies in the security mode. The ANN module is a good candidate technology for an automatic verification of the security condition. *Sec-HGS-Sched* algorithm needs also the shortest time measured in the genetic epochs for the detection of the best schedules, which is illustrated in Table 5.9. This algorithm is the best in 7 instances, and the execution time for this method may be reduced in 25%–59% in the static case and in 21%–40% in the dynamic case.

Table 5.6 Average values of *Makespan* for single-population, multi-level and hybrid genetic schedulers [$\pm s.d.$], ($s.d.$ = standard deviation)

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-ANN	4208842.037 [± 210505.265]	4216980.163 [± 249225.887]	4309539.605 [± 263233.057]	4399950.825 [± 290453.201]
Sec-HGS-Sched	**3902040.474 [± 249630.764]**	**4051566.475 [± 319691.981]**	**4101943.296 [± 308590.795]**	**414387056.050 [± 2664631.423]**
Sec-IGA	4000936.859 [± 271909.245]	4208675.544 [±292686.570]	4245347.850 [± 328969.468]	4377434.150 [± 339217.338]
Sec-(GA+TS)	4070923.243 [± 282963.771]	4195886.584 [± 249817.482]	4278491.285 [± 262374.619]	4400502.382 [± 281474.189]
		Dynamic Instances		
GA-SS-ANN	4141538.885 [± 24798859.145]	4212439.475 [± 342459.080]	4232327.490 [± 333199.727]	4364692.950 [± 339043.674]
Sec-HGS-Sched	**3971502.411 [± 259973.626]**	**3991503.974 [± 321385.198]**	**4198873.263 [± 251572.072]**	**4227569.385 [± 281680.755]**
Sec-IGA	4064692.950 [± 339043.674]	4162170.950 [± 302537.087]	4202452.157 [± 277217.723]	4315327.490 [± 344120.912]
Sec-(GA+TS)	4039043.535 [± 281858.929]	4068783.648 [± 253899.771]	4230791.746 [± 290132.215]	4263913.826 [± 294304.036]

Table 5.7 Average values of *Mean_Folwtime* for single-population, multi-level and hybrid genetic [$\pm s.d.$], ($s.d.$ = standard deviation)

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-ANN	1098725220.445 [± 148984029.042]	2261958805.835 [± 196213971.853]	4395864089.470 [± 103819795.484]	8705728350.062 [± 179128466.164]
Sec-HGS-Sched	**1065676446.564 [± 101692277.056]**	2143359732.256 [± 211454784.794]	4294563557.141 [± 373883906.206]	**8514397268.110 [± 602503134.100]**
Sec-IGA	1085575340.426 [± 110993632.105]	**2138208217.698 [± 221258711.190]**	**4236077792.436 [± 404456270.115]**	8593447951.179 [± 551754278.966]
Sec-(GA+TS)	1102225326.145 [± 197874153.696]	2199643747.642 [± 189693364.444]	4296055243.299 [± 386740590.285]	8608732539.636 [± 504003787.972]
		Dynamic Instances		
GA-SS-ANN	1163342728.245 [± 136548966.434]	2161846250.347 [± 272493690.708]	4322245472.632 [± 533180226.552]	8734534678.245 [± 635468708.749]
Sec-HGS-Sched	**1100334164.177 [± 181318391.192]**	**2113783653.774 [± 22486203.090]**	**4269654378.495 [± 547345211.754]**	8701108455.913 **[± 779952031.937]**
Sec-IGA	1198943746.287 [± 139503645.521]	2198965387.563 [± 221434723.381]	4308567534.205 [± 50953994.605]	**8666386800.606 [± 884803516.367]**
Sec-(GA+TS)	1189239424.349 [± 132687197.083]	2197268532.324 [± 167974536.172]	4320061767.548 [± 468802204.277]	8800435684.376 [± 800470337.071]

Table 5.8 Average values of failure rate $Fail_r$ parameter for single-population, multi-level and hybrid genetic schedulers [$\pm s.d.$], ($s.d.$ = standard deviation)

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-ANN	3.993% [± 0.98]	4.089% [± 1.56]	8.436% [± 1.67]	8.736% [± 2.09]
Sec-HGS-Sched	**3.522% [± 1.04]**	**4.011% [± 1.33]**	5.342% [± 0.98]	**5.328% [± 1.02]**
Sec-IGA	4.167% [± 0.98]	4.324% [± 1.26]	5.944% [± 1.89]	6.035% [± 2.23]
Sec-(GA+TS)	4.378% [± 0.92]	**5.016%** [± 1.05]	6.223% **[± 1.35]**	6.927% [± 1.56]
		Dynamic Instances		
GA-SS-ANN	4.880% [± 0.98]	6.097% [± 1.62]	7.456% [± 1.32]	7.026% [± 2.11]
Sec-HGS-Sched	**3.116% [± 0.80]**	**3.994% [± 0.93]**	**4.250% [± 0.99]**	**4.845 [± 1.14]**
Sec-IGA	4.030% [± 0.90]	4.951% [± 0.81]	5.016% [± 1.05]	5.136% [± 1.36]
Sec-(GA+TS)	4.93% [± 1.18]	6.93% [± 0.99]	6.327% [± 0.94]	6.001% [± 2.10]

Table 5.9 The number of genetic epochs necessary for the generation of the best solutions found by single-population, hybrid and multi-level schedulers

Strategy	Small	Medium	Large	Very Large
		Static Instances		
GA-SS-ANN	1722 (67.26%)	2733 (53.37%)	7992 (78.04%)	17739 (86.61%)
Sec-HGS-Sched	**1645 (64.25%)**	**2118 (41.24%)**	**6995 (63.60%)**	**15355 (74.97%)**
Sec-IGA	1788 (69.84%)	2475 (48.33%)	7044(68.78%)	16227 (79.22%)
Sec-(GA+TS)	1702 (66.48%)	2688 (52.50%)	7110 (69.42%)	15998 (78.11%)
		Dynamic Instances		
GA-SS-ANN	1522 (60.60%)	3021 (59.23%)	8010 (78.52%)	17830 (87.06%)
Sec-HGS-Sched	**1505 (59,93%)**	2935 (57.54%)	**7877 (77.22%)**	**16285 (79.51%)**
Sec-IGA	1612 (64.19%)	**2924 (57.33%)**	8054 (78.96%)	17922 (87.45%)
Sec-(GA+TS)	1578 (62.84%)	2989 (58.60%)	8035 (78.77%)	17911 (87.39%)

5.6 Conclusions

This chapter addressed the problem of the integration of the security mechanisms as additional criterion in the grid scheduling. Artificial Neural Network (ANN) was successfully implemented as the support mechanism for risk resilient genetic-based schedulers. The high effectiveness of this support was demonstrated by a comparison of the results of the performance of various GA-based schedulers in risky and secure scheduling scenarios. The proposed neural networks model seems to be a good solution for automatic monitoring of the grid system performance, but also a candidate technology for supporting the decision processes of the grid users and managers. In fact, all grid users working at different levels of the system, may specify their own strategies and preferences related to the security aspects in the scheduling process. In online scheduling, the users decisions are usually supported by the fuzzy-based online learning methodologies [110]. In batch scheduling the users' strategies and actions may be modelled by using the game-theory, as it is presented in the next chapter.