# Chapter 6
# Game-Theoretical Models of the Grid User Decisions in Security-Assured Scheduling: Basic Principles and Heuristic-Based Solutions

**Abstract.** This chapter presents two non-cooperative game approaches, namely the *symmetric non-zero sum* game and *asymmetric Stackelberg* game, for modelling the grid users' behavior. These models allow to illustrate new scenarios in scheduling and resource allocation problems, such as asymmetric users' relations, security and reliability restrictions in computational grids (CGs). Four GA-based hybrid schedulers are implemented for the approximation of the equilibrium states of both games. The proposed hybrid resolution methods are empirically evaluated through the grid simulator under the heterogeneity, security, large-scale and dynamics conditions.

## 6.1 Introduction

The security scheduling conditions defined in the previous chapter, may not be specified just by the analysis of the type of the applications submitted to grid, some local access policies to the grid resources, or the behavior and system security attributes defined in Sec. 5.3. Different types of the grid users may address their own individual requirements for the secure assignments of their tasks to the most trustful resources. In such a case the scheduling problem may be formulated as the decision problem of the grid users working at the different levels of the grid system.

In CGs the system management techniques must be able to group, predict, and classify different sets of rules, configuration directives, and environmental conditions. This management model must effectively deal with uncertainties in system information, that may be incomplete, imprecise, fragmentary, or overloading to control specific constituents and objects within intricate configurations. Decision scenarios ought to be outlined assuming partial visibility of environmental conditions, user heterogeneity, and resource dynamism in order to determine and select adequate evaluation criteria and assignment scores to render a final integrated decision result.

In many decision-making problems, most of the information is provided by humans, which is inherently non-numeric. Partial evaluations, preferences, weights are expressed linguistically. The evident role of fuzzy sets in decision-making and

J. Kołodziej: Evolutionary Hierarchical Multi-Criteria Metaheuristics, SCI 419, pp. 113–135.
springerlink.com 

associated important processes such as consensus building, is well documented in the literature [118], [119], [69]. However, in large-scale grid systems all the users' information must be analyzed and interpreted in a short time, and special users' preferences must be taken into account. The fuzzy rules may be then wrongly interpreted by the system management components, or filtered in order to design the optimal strategies for minimizing the scheduling costs.

Game-theoretical models may be considered as alternative solutions for large-scale decision problems in highly parametrized heterogeneous environments. All scheduling criteria can be aggregated and defined as cumulative users' cost or pay-off functions, which makes the game models very useful in the analysis of the various users' strategies in the resource allocation process. Game-based models combined with the economic theory can capture many realistic scenarios in computational markets, computational auctions, grid and P2P systems as well as security and information markets. An important challenge in using game-theoretic models for grid scheduling and resource management is the large size scale of the grid system and the fact that resources cross different administrative domains. The grid game players should behave rationally, pursue well-defined objective functions (cost or pay-off functions), and react fast to the other players' actions and decisions.

This work is based on the results presented in [85], where the preliminary versions of the symmetric and asymmetric non-cooperative grid users games were defined for the purpose of illustrating the users strategies in the security-aware scheduling. This chapter summarizes those results and presents the formal models of general symmetric and asymmetric Stackelbelg games. These models are based on the premise about the users' behavior in a realistic large-scale grid, where users, usually independently of each to another, submit their tasks/applications to the grid system. Additionally, in the Stackelberg game, one player (user) is acting as a Leader with a privileged access to resources. This Leader assigns his tasks first, and the rest of the users (Followers) react rationally to the Leader's actions. The Followers do not cooperate with each other, but their decisions depend on the Leader's action. This model illustrates very well the real-life situation, where the roles of the users are in fact asymmetric with regard to their access rights and usage of resources. It must also be noted that in many economical models the sellers and buyers stand in asymmetric positions as well. Having a control over large resource pools and maintaining the large fraction of the task batch for scheduling can be the reasons of having some privileges in the resource access, or in the setting of the reasonable resource utilization pricing policies.

The users cost functions in the game are interpreted as the cummulative cost of the secure execution of their tasks and the costs of the utilization of resources. The cumulative cost function specified for the whole game is optimized at global and local (users') levels, through four genetic-based hybrid meta-heuristics, which combine Genetic Algorithm (GAs) and modified Minimum Completion Time (MCT) method.

Two scheduling scenarios are considered in in this work, namely *risky* and *secure* mode. In the former, security conditions are ignored by the users by allocating their tasks to all available machines, independently of the trusted levels. In the later, users

allocate tasks to available machines assuring task security demands. It should be noted that in task scheduling the definition of the security demand can be two-fold: (a) tasks can have security demands on resources to be allocated at and (b) resources can have security demands on tasks to be assigned to them. This work is focused on the condition (a) of security requirements.

The proposed models were evaluated under the heterogeneity, the large scale and dynamics conditions using the *Sim-G-Batch simulator*. The relative performance of four hybrid schedulers are measured by the makespan and flowtime metrics. However, the main aim of the empirical analysis is to compare the effectiveness of the game models in the reduction of the scheduling costs in the secure scenario, and the results achieved by the best single-population grid scheduler generated in Chapter 5 for the secure scheduling with the ANN support (see Sec. 5.5.3.1).

## 6.2 Users' Behavior Models in Grid Scheduling

The classifications of the grid scheduling problems presented in Sec. 1.3.2 in Chapter 1 do not span over the analysis of the relations and behavior of the grid users at different systems levels (see Fig. 1.3). Three basic models of grid users' relation in grid scheduling processes can be defined as follows:

- **Cooperativeness:** In this case the users can form a coalition to plan in advance their actions;
- **Non-cooperativeness:** In this scenario the users act independently of one another;
- **Semi-cooperativeness:** In this model each user can select a partner for the cooperation.

The analysis of the above mentioned relations of the users is used for the specification of the generic models of the following three types of grid user games, namely *non-cooperative*, *cooperative* and *semi-cooperative* games:

- In **non-cooperative game** the players act independently of each other. This model is based on the premise about the users' behavior in realistic grids, where cooperation is difficult in large-scale system, and grid users submit their tasks independently. Also the resource owners act selfishly in order to maximize the resource utilization and to execute the tasks from the local users.
- In **cooperative game** the players can form a coalition to plan their future actions. This model is useful for the intra-site grid negotiations, where the local job dispatchers can define the joint "execution capabilities" parameters for the clusters of the grid sites and declare them to the global scheduler.
- In **semi-cooperative game** each player can choose (randomly) another player for cooperation. This game is usually proposed as a multi-round auction to incorporate the task rescheduling.

The solution of each of those games is an equilibrium state, in which each player holds correct expectations concerning the other players' behavior (see [75] for the detailed analysis).

The users can have different privileges to the resources, resulting in the examination of the following two scenarios:

- **Symmetric scenario.** In this case there are no special privileges in the resource usage for the grid users.
- **Asymmetric scenario.** In this case there is a privileged user (Leader), who can have full access to resources as opposed to the rest of users who can be granted only limited access to resources. The Leader could also be the owner of a large portion of the task pool, as it is reasonable to allocate first his tasks at best resources in the system.

The game models presented in this chapter are based on the non-cooperative scenario in symmetric and asymmetric modes.

## 6.3 Symmetric and Asymmetric Games of Independent Grid Users

One of the main benefits of the game-based scheduling and resource management in CGs is that it enables a scalability and personalization of the decision-making processes of grid users and resource owners. Due to the sheer scale of grid systems, the non-cooperative game is a potential model for integrating security and resource reliability requirements in grid scheduling. This section presents two different general scenarios of the non-cooperative grid users behaviors, namely *symmetric* and *asymmetric* strategic game models.

### *6.3.1 Non-cooperative Symmetric Game*

Let us denote by *Play* the number of grid users (players). The total number of tasks $n \in N$ in a given batch can be expressed as the sum of numbers of tasks submitted by all users, i.e.

$$n = \sum_{a=1}^{Play} k_a, \tag{6.1}$$

where $k_a$ is the number of tasks of the user $a = 1, \ldots, Play$.

Each player $a$ controls his strategic variables defined as the following *user's strategy vector* :

$$Pl_a = \left[ j_{\widehat{(k_{(a-1)}+1)}}, \ldots, j_{\widehat{(k_{(a-1)}+k_a)}} \right] \tag{6.2}$$

where $\widehat{k_{(a-1)}} = k_1 + \ldots + k_{(a-1)}$

The schedules can be then expressed by the following vectors of the users' parameters:

$$S = \left[ i_1^1, \ldots, i_{k_1}^1, \ldots, i^a_{\widehat{(k_{(a-1)}+1)}}, \ldots, i^a_{\widehat{(k_{(a-1)}+k_a)}}, \ldots\ldots i^{Play}_{k_{Play}} \right], \tag{6.3}$$

in the direct representation, and

$$Sch = [Pl_1, \ldots, Pl_{Play}], \tag{6.4}$$

in permutation-based representation (see Chapter 2, Sec. 2.2.1 for details on the schedules representations).

In *symmetric* non-cooperative users' game the privileges to the resources are the same for all users. Each user tries to choose an optimal strategy for the assignment of his tasks to machines in order to minimize his cost of tasks scheduling and, as the results, also the overall scheduling costs. An illustrative example of the symmetric game can be a scheduling scenario in which each player submits an equal amount of tasks, i.e. $k = k_1 = k_2 = , \ldots, = k_{Play}$. It means that in such a case the total number of tasks in the batch can be calculated in the following way: $n = Play \cdot k$.

**Definition 6.1.** The symmetric grid users' non-cooperative game can be defined as a tuple

$G_{Play} = (Play; \{J_a\}_{a=1,\ldots Play}; \{Q_a\}_{a=1,\ldots,Play})$, where:

- $Play$ is the number of grid users;
- $\{J_1, \ldots, J_{Play}\}$; are the sets of users' strategies;
- $\{Q_1, \ldots, Q_{Play}\}; Q_a : J_1 \times \ldots \times J_{Play} \rightarrow \mathbb{R}; \forall_{a=1,\ldots,Play}$ is the set of users' cost functions.

The users' strategy vectors $Pl_a$ are the elements of the strategy spaces $J_l = J_{((a-1)\cdot k+1)} \times \ldots \times J_{(a \cdot k)}$ specified for each user $a, (a = 1, \ldots, Play)$. The cost of playing the game calculated for a particular user $a$ is defined as the cost of scheduling of his tasks (or the user's cost function) and is denoted by $Q_a$. The players try to minimize simultaneously their cost functions $Q_a$ in the game.

**Definition 6.2.** A multi-vector $(\widehat{Pl_1}, \ldots, \widehat{Pl_{Play}})$ of strategies is called **an equilibrium state (point)** of the game if :

$$\begin{aligned} &Q_a\left(\widehat{PL_1}, \ldots, \widehat{PL_{Play}}\right) = \\ &= \min_{Pl_1 \in J_1} Q_a\left(\widehat{Pl_1}, \ldots, \widehat{PL_{(a-1)}}, PL_a, \widehat{Pl_{(a+1)}}, \ldots, \widehat{Pl_{Play}}\right) \end{aligned} \tag{6.5}$$

for all $a = 1, \ldots, Play$.

The formulas of $Q_a$ functions are specified for the permutation-based representation of the schedules because of the simpler notation used in the Eq. (6.4). Those procedures can be easily transformed into the direct representation by substituting the $\left(\widehat{PL_1}, \ldots, \widehat{PL_{Play}}\right)$ vector by the vector defined in Eq. (6.3). The equilibrium point can be interpreted as a steady state of the a strategic game, in which each player holds correct expectations concerning the other players behavior[1]. If the strategies

[1] In the case of continuous players' cost functions the equilibrium state of the game is called the *Nash equilibrium* [39].

chosen by all players are equilibrium points, no player is interested in changing his strategy.

To be a solution of the grid users', the game the equilibrium point should be additionally Pareto-optimal [139, 117]. In this chapter we consider the non-zero sum games[2], for which the equilibrium points are the results of minimization of a *multi-cost game function* $Q$ defined as follows.

Let us denote by $minQ_a, (a = 1, \ldots Play)$, the minimal value of the function $Q_a$ calculated for each user $a$, that is to say:

$$minQ_a = \min_{PL_a \in J_a} \{Q_a(PL_1, ..., PL_{Play})\}. \tag{6.6}$$

The results of the global minimization of the following *game multi-cost* function $Q : J_1 \times \cdots \times J_{Play} \to \mathbb{R}$:

$$Q(PL_1, ...., PL_{Play}) = \sum_{a=1}^{Play} \frac{1}{Play} \left(Q_a(PL_1, ..., PL_{Play}) - minQ_a\right), \tag{6.7}$$

is an equilibrium state of non-cooperative non-zero sum symmetric game of the grid users, which satisfies the condition of the Pareto-optimality [117][3].

### 6.3.2 Asymmetric Scenario – Stackelberg Game

The symmetric games are quite simple for the implementation and well studied for many high-performance computing approaches. However, the symmetric scenario may not be a good model of the realistic users' relations. Due to the cross-domain access, authorization and resource management features of the grid system, the grid users have different access policies to the resources and they stand in an asymmetric position with regard to resource usage privileges. The asymmetric behavior of grid users directly impacts the results of the scheduling process.

A Stackelberg game is the simplest model for illustrating the asymmetric scenario of the behavior of non-cooperative grid users. In this game one privileged user acts as a *Leader*, and the rest of players (users) are his *Followers*.

The Stackelberg games have been well-studied in the game theory literature (see, e.g. [10]). Roughgarden [126] defined a Stackelberg game model for scheduling tasks on a set of machines with load-dependent latencies in order to minimize the total latency of the system.

The following examples illustrate some real-life grid scenarios, to which the Stackelberg game model can be applied:

[2] In this scenario the strategies of the players are not opposite, i.e. the sum or the values of all players cost functions $Q_a$ is not 0.

[3] In fact, the function $Q$ is a special case of the weighed or weighed distance $L^p$ metric function with $p = 1$. The values of all $Q_a$ functions are non-negative, and the weight coordinates are strictly positive, which means that the global solutions of the problem defined in Eq. (6.7)are Pareto-optimal [142].

- There is a privileged grid user (Leader), who can have the full access to resources as opposed to the other users with limited access to resources.
- Some tasks can have critical deadlines (especially in online scheduling) and they can be sent by the Leader to the meta-broker with a request to allocate them first.
- Considering a batch of tasks, the Leader can be the owner of a large portion of the tasks in the batch; therefore it might be reasonable to allocate his tasks to the best resources in the system.
- Some tasks could have security requirements. Therefore the Leader can send such an information and all security parameters to the scheduler or directly to the meta-broker requesting to allocate them in the most trustful resources (secure machines).
- Tasks submitting to a grid system could be varied in their needs for computational resources. Some of them could be atomic tasks generated by compound tasks while the others could be just monolithic applications. The high degree of heterogeneity of tasks usually has a great impact on the grid system's performance. In such a scenario the Leader could create a small batch of the most time consuming tasks as the backlog set of grid applications, in order to "balance" the computational loads of machines during the scheduling. These tasks would be sent to the meta-broker requesting to allocate them first.

Formally the two-level Stackelberg game of the grid users can be defined in the following way:

- **Leader's Level: Leader's action I -** Leader chooses his initial strategy $\widetilde{Pl_1} = [\widetilde{j_1}, \ldots, \widetilde{j_{k_1}}]$, where $k_1$ denotes the number of tasks submitted by the Leader.
- **Followers' Level: Followers' action -** Followers minimize simultaneously their cost functions relative to the Leader's choice:

$$\begin{cases} Pl_2^{Fol} = arg\min_{(Pl_2 \in J_2)} \{Q_2(\widetilde{Pl_1}, Pl_2, \ldots, Pl_{Play})\} \\ \vdots \\ Pl_{Play}^{Fol} = arg\min_{(Pl_{Play}) \in J_{Play}} \{Q_{Play}(\widetilde{Pl_1}, \ldots, Pl_{Play})\} \end{cases} \tag{6.8}$$

  where $J_1$ is the set of the Leader's strategies and $Q_a$ is the cost function of the user $a$ defined as in the symmetric case in Eq. (6.12). Let us denote by $Pl^{Fol} = \left[\widetilde{Pl_1}, Pl_2^{Fol}, \ldots, Pl_{Play}^{Fol}\right]$ a *Followers' Vector*, which is interpreted as the result of the Followers' action.
- **Leader's Level: Leader's action II -** Leader updates his strategy by minimizing his cost function $Q_1$ (see also Eq. (6.12)) taking into account the result of Followers' actions. The following vector $Pl^G = \left[Pl^{Lead}, Pl_2^{Fol}, \ldots, Pl_{Play}^{Fol}\right]$, where:

$$Pl^{Lead} = arg \min_{(Pl_1 \in J_1)} Q_1\left(Pl_1, Pl_2^{Fol}, \ldots, Pl_{Play}^{Fol}\right) \tag{6.9}$$

  is a solution of the whole game.

It has to be noted that the Followers can play an "ordinary" non-cooperative symmetric game, but they must know the Leader's action first. The game multi-cost function $Q$ in this case can be defined in the following way:

$$Q_{Stac} = \frac{1}{Play} Q_1 + Q_{Fol}; \tag{6.10}$$

where $Q_1$ is the Leader's cost function and

$$Q_{Fol} := \frac{Play - 1}{Play} \sum_{a=2}^{Play} Q_a; \tag{6.11}$$

is a *Followers' cost function*. An optimal solution of the whole game is called *Stackelberg Equilibrium*.

#### 6.3.2.1 Users' Cost Functions in Security-Aware Scheduling

In conventional grid scheduling with the typical scheduling objective functions such as makespan and flowtime, the users' costs of scheduling their tasks are limited to the costs of tasks execution. In utility grids, there are the resource utility functions that must be specified for the calculation of the resource utilization cost [50]. In security-assured scheduling some additional costs must be considered. The users have to "pay" an additional "fee" for the secure allocation of their tasks in the machines. In this work all of those costs are integrated into cumulative-cost functions $Q_a, a \in \{1, \ldots, Play\}$, defined separately for each grid user as the weighed sum of the following three components:

$$Q_a(S) = Q_a^{(ex)}(S) + Q_a^{(util)}(S) + Q_a^{(sec)}(S), \tag{6.12}$$

where:

- $Q_a^{(ex)}$ indicates the user's task execution cost ,
- $Q_a^{(util)}$ denotes the resource utilization cost , and
- $Q_a^{(sec)}$ is the cost of security-assured allocation of the user tasks .

In this work the ETC Matrix model (see Chapter 2) is used for the specification of all cost functions for the users.

### *6.3.3 Task Execution Cost*

The total cost of execution of the user's tasks can be calculated as an average completion time of his tasks on machines, to which they are allocated[4]. In terms of the

[4] The values of all components of the users game cost functions, i.e. $Q_a^{(ex)}$, $Q_a^{(util)}$ and $Q_a^{(sec)}$ functions, are scaled to get the all values in the same range.

completion times of machines (see Chapter 2, Eq. (2.13)) the function $Q_a^{(ex)}$ can be defined using the following formula:

$$Q_a^{(ex)} = \frac{\sum_{j=k_{a-1}+1}^{k_a} completion[j][i]}{completion_m \cdot k_a}, \tag{6.13}$$

where $completions[j][i]$ denotes the completion time of a task $j$ on a given machine machine $i$ and and it is calculated in the following way:

$$completions[j][i] = ETC[j][i] + ready[i]. \tag{6.14}$$

In Eq. (6.13), the $completion_m$ indicates the maximal completion time of all tasks submitted by the user $a$, that is to say:

$$completion_m = \max_{j=k_{a-1}+1,\ldots,k_a} completion[j][i]. \tag{6.15}$$

### 6.3.4 Resource Utilization Cost

The grid user's utility function is usually defined as a cost of buying free CPU cycles [50]. In this work the utilization cost paid by the user $a$ is calculated as a "portion" of the average idle time of machines on which his tasks are executed. This cost depends on the completion times of the user's tasks. The utility function $Q_a^{(util)}$ is defined as follows:

$$Q_a^{(util)} = \sum_{i \in machines(a)} \left(1 - \frac{Completion_{(a)}[i]}{C_{max}}\right) \cdot Idle\_Factor[i] \tag{6.16}$$

where $machines(a)$ denotes the set of machines, to which all tasks of the user $a$ are assigned and $C_{max}$ refers to the makespan. The completion time of a given machine $i \in machines(a)$, denoted by $Completion_{(a)}[i]$, is calculated in the following way:

$$Completion_{(a)}[i] = ready[i] + \sum_{\substack{j \in N: \\ S[j]=i}} ETC[j][i] \tag{6.17}$$

where $S[j]$ is the value of $j$-th coordinate in a given schedule vector $S$ (or $Sch$ – both implementations of the schedules may be used in Eq. (6.17). The following expression:

$$\left(1 - \frac{Completion_{(a)}[i]}{C_{max}}\right) \cdot Idle\_Factor[i], \tag{6.18}$$

in Eq. (6.16) is interpreted as an idle time of machine $i$ calculated for a given user $a$. This is just a "portion" of the total idle time of machine $i$, and it is proportional

to the time of execution of all tasks of the user $a$ assigned to this machine. This proportion is specified by the coefficient $Idle\_Factor[i]$ in the following way:

$$Idle\_Factor[i] = \frac{\sum_{j \in Tasks_{(a)}[i]} ETC[j][i]}{Completion_{(a)}[i]} \tag{6.19}$$

where $Tasks_{(a)}[i]$ is the set of the tasks of the user $a$ assigned to the machine $i$.

It follows from Eq. (6.16) that the utilization cost is minimal in the case of allocation of the user tasks to machines with the maximal completion times.

### 6.3.5 Security-Assurance Cost

The security-assurance cost of scheduling the tasks of the user $a$, denoted by $Q_a^{(sec)}$ in Eq. (6.12, depends on the scheduling strategy and the result of the verification of security condition by the trust manager. The manager must analyze the security demand $SD$ and trust level $TL$ vectors for tasks and machines and the *Machine Failure Probability* matrix $Pr_f = \left[Pr_f[j][i]\right]_{n \times m}$ must be specified in the similar way as in Eq. (5.2), that is to say:

$$Pr_f[j][i] = \begin{cases} 0 & , sd_j \leq tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & , sd_j > tl_i \end{cases} \tag{6.20}$$

where $\alpha$ is the failure coefficient and $sd_j$ and $tl_i$ are the security demand and trust level parameters for task $j$ and machine $i$.

Similarly to Sec. 5.3.1 in Chapter 5, two different scheduling strategies can be considered, namely *Risky* and *Secure* modes, in order to illustrate the various users' and grid managers' decisions. The formulas for calculating the security cost for the users are based on the formulas for the completion times of machines and flowtime in Eq. (5.3)– (5.6).

In the **Risky mode** all risky and failing conditions are ignored by the users. In this case the "security" components of the functions $Q_a$ are in fact not calculated, i.e. $Q_a^{(sec)} = 0, \forall a = 1, \ldots, Play$. However, some machines may fail during the tasks execution because of too restrictive security requirements and rescheduling procedure of those tasks must be activated. Therefore the security cost in this case is calculated as follows:

$$Q_l^{(sec)}[ris] = \sum_{j \in Res(a)} \frac{P_f[j][i] \cdot ETC[j][i]}{(ETC)_{m(a)} \cdot \sharp(Res(a))}, \tag{6.21}$$

where $Res(a)$ is the set of the tasks of the user $a$ which must be rescheduled, and $(ETC)_{m(a)}$ is the (expected) maximal computation time of the tasks of the user $a$ in a considered schedule, i.e.:

$$(ETC)_{m(a)} = \max_{\substack{j \in Task(a) \\ i \in machines(a)}} ETC[j][i]. \tag{6.22}$$

In Eq. (6.22), $Task(a)$ denotes the set of the tasks of the user $a$ and $machines(a)$ is the set of the machines to which the user tasks are mapped in a considered schedule.

In **Secure mode** the users must pay the cost of the verification of the security condition for his tasks (see Sec. 5.3). The cost of possible failures of machines during the tasks executions are calculated as the products of the failure probabilities and the expected times of computation of the tasks on the inaccessible machines. The secure cost function $Q_a^{(sec)}$ in this case is defined as follows:

$$Q_a^{(sec)}[secure] = \sum_{j=k_{a-1}+1}^{k_a} \frac{P_f[j][i] \cdot ETC[j][i]}{(ETC)_{m(a)} \cdot k_a}, \tag{6.23}$$

The security-assurance cost expressed as $Q_a^{(sec)}[secure]$ for each grid user is minimized. It means that each user tries to allocate his tasks in the most trustful resources and the values of task failure probabilities $P_f[j][i]$ should be minimal.

## 6.4 Solving the Grid Users Games

The problem of solving the finite strategic game remains challenging especially in real-life approaches. In order to compute the values of the game cost functions $Q$ defined in Eqs. (6.7) and (6.10), the cost functions of all players must be first minimized. Therefore the problem of the minimization of $Q$ function can be defined as a hierarchical procedure presented in Fig. 6.1. This procedure is composed of two cooperating modules: **Global Module**, in which the values of the function $Q$ are calculated and optimized, and the **Players' Module** - which solves the local level problems of the minimization of the users' cost functions $Q_a$.

The communication procedure between *Global* and *Players' Modules* can be defined as follows: Let us denote by $S^{(0)}$ an initial schedule generated in the Global Module, i.e. $S^{(0)} = [Pl_1^{(0)}, \ldots, Pl_{Play}^{(0)}]$, where $Pl_a^{(0)}$ is the initial strategy vector of the

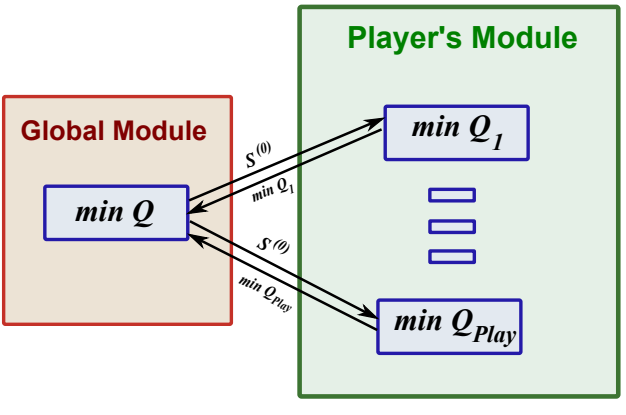


**Fig. 6.1** Hierarchical procedure of solving non-cooperative symmetric game of grid users

user $a$ (see Eq. (6.2)). Vector $S^{(0)}$ is replicated and its copies are sent to the *Players' Module* - one copy per user. Then, each user independently optimizes his game cost function[5] by changing the allocations of just his own tasks. As the result of this minimization, the optimal values of the $Q_a$ cost functions are calculated:

$$\begin{cases} minQ_a{}^{(0)} = \min_{(Pl_1 \in J_1)} Q_1\left(Pl_1, Pl_2^{(0)}, \ldots, Pl_{Play}^{(0)}\right) \\ \vdots \\ minQ_{Play}{}^{(0)} = \min_{(Pl_{Play} \in J_{Play})} Q_{Play}\left(Pl_1^{(0)}, \ldots, Pl_{Play-1}^{(0)}, Pl_{Play}\right) \end{cases} \tag{6.24}$$

These values are sent back to the **Global Module**, where the objective function for the whole game $Q$ is calculated for the schedule $S^{(0)}$.

In the case of Stackelberg game the *Global Module* plays the role of the *Leader's* component and the *Player's Module* – the *Follower's* procedure, as it is presented in Fig. 6.2.

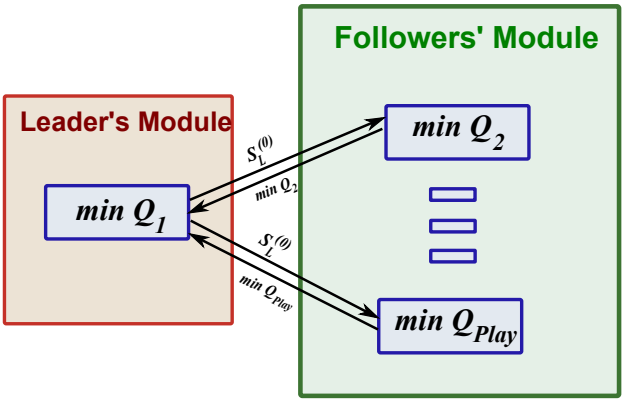


**Fig. 6.2** Hierarchical procedure of solving Stackelberg game

However, in this case the schedule vector $S_L^{(0)}$ before its replication and sending to the Followers, is partially 'fulfilled' by the Leader. The Leader makes his preliminary assignments and send the incomplete schedule vectors to the Followers.

### 6.4.1 Genetic Hybrid Metaheuristic Solvers

Similar to the empirical analysis presented in the previous chapters, and due to multiple constraints and different preferences of the grid users, genetic-based heuristic approaches seem to be the best candidate methodologies for solving the users'

[5] Note that the users costs optimization in the **Players' Module** can be implemented as a parallel multi-threaded procedure, which can speed-up the whole process.

games. However, in this case the main framework of the scheduler must be extended by the hybridization of genetic algorithm (GA) working in the **Global/Leader's Module** with some other heuristic method implemented in the **Players'/Followers' Module**.

Four hybrid GA-based schedulers have been defined for solving the symmetric and asymmetric grid users' games. The combinations of the heuristic components of these hybrids are presented in Table 6.1.

**Table 6.1** Hybrid meta-heuristics for risky and secure-assured scheduling

| Meta-heuristic | **Global/Leader's Module** | **Players'/Followers' Module** |
|---|---|---|
| **RGA-GA** | RGA | PGA |
| **RGA-PMCT** | RGA | PMCT |
| **SGA-GA** | SGA | PGA |
| **SGA-PMCT** | SGA | PMCT |

The GA-based meta-heuristics may work as global and local optimizers in the **Global/Leader** and **Players'/Followers'** Modules. Each hybrid algorithm is defined as a combination of two methods, namely *Risky Genetic Algorithm (RGA)* and *Secure Genetic Algorithm (SGA)*– in the **Global Module**; and two local level optimizers, namely *Player's Genetic Algorithm (PGA)* and *Player's Minimum Completion Time (PMCT)*– in the **Players' Module**. It can be observed that, in fact, it is not necessary to replicate the whole population form the Global or Leader's Module to the Players'/Followers' Module. For each player independently, just the changes in machine completion times must be updated. Therefore the general procedure of Players'/Followers algorithm may be defined as follows:

**Algorithm 3.** The optimization procedure in **Player's/Followers' Module**

1: Send the *ready_times* vectors to the individual players;
2: Individual players compute the $MinQ_a$ values;
3: Receive the $MinQ_a$ values from the individual players;
4: Send the $MinQ_a$ values to **Global Module**;

### Schedulers Implemented in Global and Leader's Module

The generic template of the main GA-based engine of the hybrid scheduler designed for solving the symmetric games is presented in Alg. 4. This template is similar to the Alg. 1 defined in Chapter 3 for HGS-Sched (see. Sec. 3.3).

**Algorithm 4.** Genetic Algorithm template

1: Generate the initial population $P^0$ of size $\mu$;
2: Send the *ready_times* vectors of the machines corresponding to the individuals of the population $P^0$ to the **Player's Module**;
3: Receive the $minQ_a$ values from the subordinate unit
4: Evaluate $P^0$;
5: **while** not termination-condition **do**
6: Select the parental pool $T^t$ of size $\lambda$; $T^t := Select(P^t)$;
7: Perform crossover procedure on pars of individuals in $T^t$ with probability $p_c$; $P_c^t := Cross(T^t)$;
8: Perform mutation procedure on individuals in $P_c^t$ with probability $p_m$; $P_m^t := Mutate(P_c^t)$;
9: Send the *ready_times* vectors of the machines corresponding to the individuals of the population $P_m^t$ to the **Player's Module**;
10: Receive the $minQ_a$ values from the Players' Module
11: Evaluate $P_m^t$ ;
12: Create a new population $P^{t+1}$ of size $\mu$ from individuals in $P^t$ and/or $P_m^t$ ;
13: $t := t+1$;
14: **end while**
15: **return** Best found individual as solution;

The main difference between *RGA* and *SGA* algorithms is the method of the evaluation of the population by using the users' cost functions $Q_a$, which is different in the Risky (RGA) and Secure (SGA) modes. The formulas of calculating the 'security' costs in both scenarios are defined in Sec. 6.3.5.

In the case of Stackelberg game the initialization procedure in the main GA algorithm is a bit different than in the symmetric scenario. The general template of the main genetic engine at the **Leader's (Global) Module** in this game is defined in Alg. 5.

**Algorithm 5.** A GA-based scheduler at the **Leader's** level

1: Generate $P^0$ containing $\mu$ "incomplete" schedules; $t = 0$;
2: Send $P^0$ to the **Followers** to complete the respective parts of all schedules in $P^0$; $P^0(F)$ is created;
3: Update the population $P^0$ according to the Followers' solutions;$P^0 := P^0(F)$;
4: Evaluate $P^0$;
5: **while** not termination-condition **do**
6: Select the parental pool $T^t$ of size $\lambda$; $T^t := Select(P^t)$;
7: Perform crossover procedures separately on Leader's and Followers'variables on pairs of individuals in $T^t(F)$ with probability $p_c$; $P_c^t := Cross(T^t)$;
8: Perform mutation procedures separately to Leader's and Followers' variables on individuals in $P_c^t$ with probability $p_m$; $P_m^t := Mutate(P_c^t)$;
9: Evaluate $P_m^t$ ;
10: Create a new population $P^{t+1}$ of size $\mu$ from individuals in $P^t$ and $P_m^t$ ; $P^{t+1} := Replace(P^t; P_m^t)$
11: $t := t+1$;
12: **end while**
13: **return** Best found individual as solution;

The process of initialization of the population in the main GA algorithm is defined as a two-step procedure. In the first step, the $P^0$ set is generated as a candidate initial population. It consists of the incomplete schedules generated by the Leader

by using one of the initialization methods for GA-based schedulers (see Chapter 3, Sec. 3.3). Each schedule from this set contains just the values of the Leader's decision variables. All those "incomplete" chromosomes are sent to the Followers' Module. The Followers complete each schedule by using one of the ad-hoc heuristics. The updated population $P^0$ is then evaluated under the game cost function $Q_{Stac}$ defined in Eq. (6.10. The crossover and mutation operations are performed separately on Leader's and Followers' decision variables. Therefore in each generation the Followers can update their own decisions (including the initial choices) according to all changes in availability of resources introduced by the Leader.

**Local Schedulers in Players' and Followers' Modules**

Two modifications of well-known grid schedulers are implemented in the Players' and Followers' Modules.

The first scheduler, called *Player's Genetic Algorithm*, is a simple extension of the classical GA-based scheduler defined in Alg. 1 applied independently for each user with the cost function $Q_a$ as the fitness measure. The genetic operations are executed on sub-schedules of the length $k_a$ labeled just by the tasks submitted by user $a$. In the implementation presented in this work the GA procedures in the Players' or Followers Modules are executed sequentially for the "queue" of users, however each algorithm may be implemented as a separate process on parallel multiprocessor machine the number of processors must be in this case the same as the number of players or followers)

The second method, called *Player's Minimum Completion Time - (PMCT)*, is the modification of *Minimum Completion Time - MCT*. In this method, a task is assigned to the machine yielding the earliest completion time (defined as the sum of *ready_time* for the machine and time of computing all tasks assigned there). The process is repeated until there remain tasks to be assigned. The template of the main mechanism of *PMCT* procedure is defined in Alg. 6.

**Algorithm 6.** *PMCT* algorithm template

```
1: Receive the population of schedules and ready_times of the machines from the Global
   Module;
2: for all Schedule in the population do
3:    Calculate the completion times of the machines in a given schedule;
4:    for all Individual user/Follower do
5:       for all User's Task/Follower's Task do
6:          Find the machine that gives minimum completion time;
7:          Assign task to its best machine;
8:          Update the machine completion time;
9:       end for
10:      Calculate the minQ_a value for a given schedule;
11:   end for
12:   Send the minQ_a values to the Global Module;
13: end for
```

## 6.5 Empirical Analysis

The main aim of the empirical evaluation of the genetic hybrid schedulers defined in the previous section is to compare the effectiveness of the game-based models in the optimization of the main scheduling objective functions, namely *Makespan* and $Mean\_Flowtime$ defined in Sec. 5.5.2, with the results achieved by the best single population $GA-CX-R-ANN$ scheduler supported by the neural network mechanism in the similar analysis in Chapter 5.

Four hybrid meta-heuristics defined in Table 6.1 have been used for solving the symmetric and asymmetric games. These methods were integrated with the *Sim-G-Batch* simulator. The experiments have been conducted on two benchmarks composed by a set of static and dynamic instances. Similarly to the empirical analysis provided in Chapters 4 and 5, four grid size scenarios are considered, namely Small, Medium, Large and Very Large grids.

The key parameters of the simulator in all experiments are the same as in Table 5.1 The parameters of *HGS-Sched* for generating the GA algorithms in **Global**, **Leader's**, **Players'** and **Followers' Modules** are defined in Table 6.2.

**Table 6.2** GA settings in the **Global/Leader's** and **Players'/Followers' Modules** for large static and dynamic benchmarks

| **Parameter** | **Global/Leader's Module** | **Players'/Followers' Module** |
|---|---|---|
| $period\_of\_metaepoch$ | $(5*(n))/10$ | $(\lceil 0.5*(n) \rceil)/10$ |
| $nb\_of\_metaepochs$ | 10 | |
| **population size** ($pop\_size$) | 60 | 20 |
| **intermediate pop.** | 48 | 14 |
| **selection method** | LinearRanking | |
| **crossover method** | CX | |
| **cross probab.** | 0.8 | 0.8 |
| **mutation method** | Rebalancing | |
| **mutation probab.** | 0.2 | |
| **initialization** | LJFR-SJFR + Random | |
| $max\_time\_to\_spend$ | 500 secs ($static$) / 800 secs ($dynamic$) | |

There are 16 players in symmetric game and 15 Followers in the Stackelberg game, and the number of the Leader's tasks is a half of the whole task batch. The coefficients of $SD$ and $TL$ vectors, and the machines reliability probabilities $P_i$ are defined as the uniformly generated fractions in the ranges [0.6;0.9], [0.3;1] and [0.85;1] respectively. The value of the failure coefficient $\lambda$ is 3.

Each experiment was repeated 30 times under the same configuration of parameters and operators.

The histograms of the average values of *Makespan* and *Mean_Flowtime* achieved by four hybrid meta-heuristics designed for solving the users games are presented in Fig. 6.3 and 6.3.

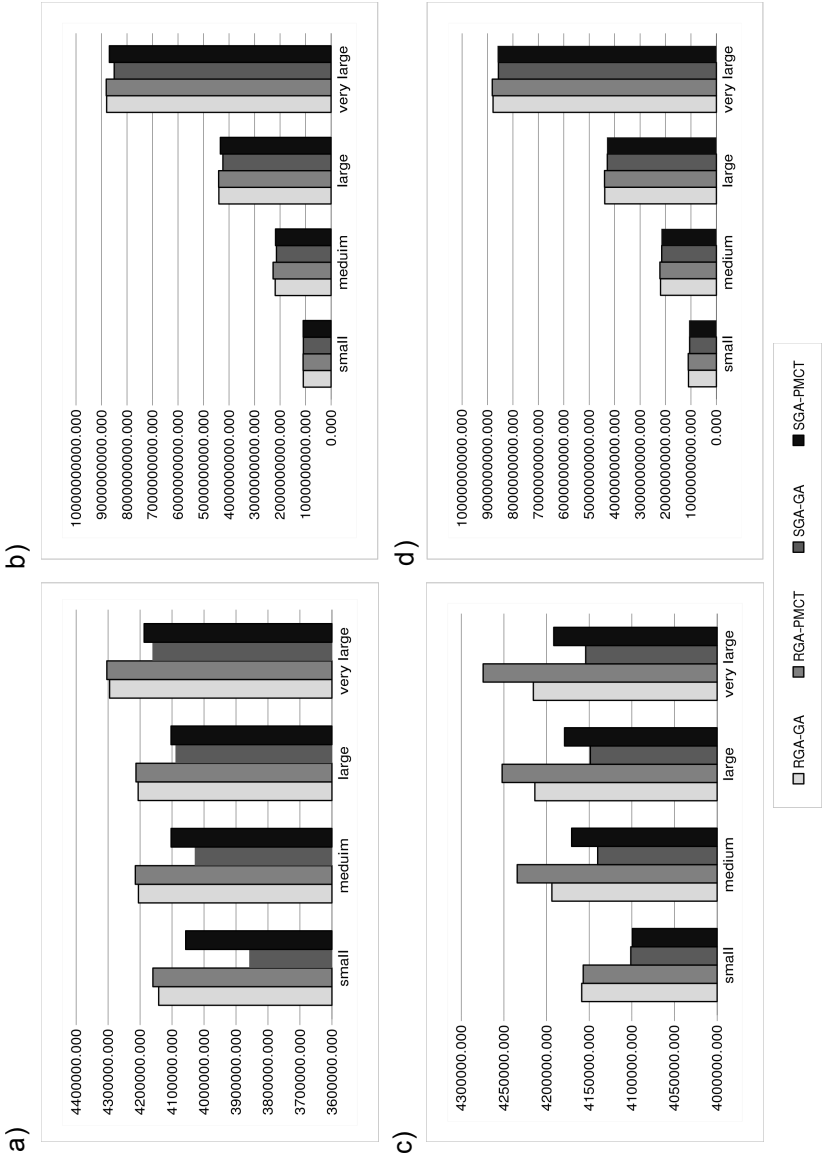


**Fig. 6.3** Experimental results for non-cooperative symmetric game: in static case - (a) average *Makespan*, (b) average *Mean_Flowtime* ; in dynamic case - (c) average *Makespan*, (d) average *Mean_Flowtime*.

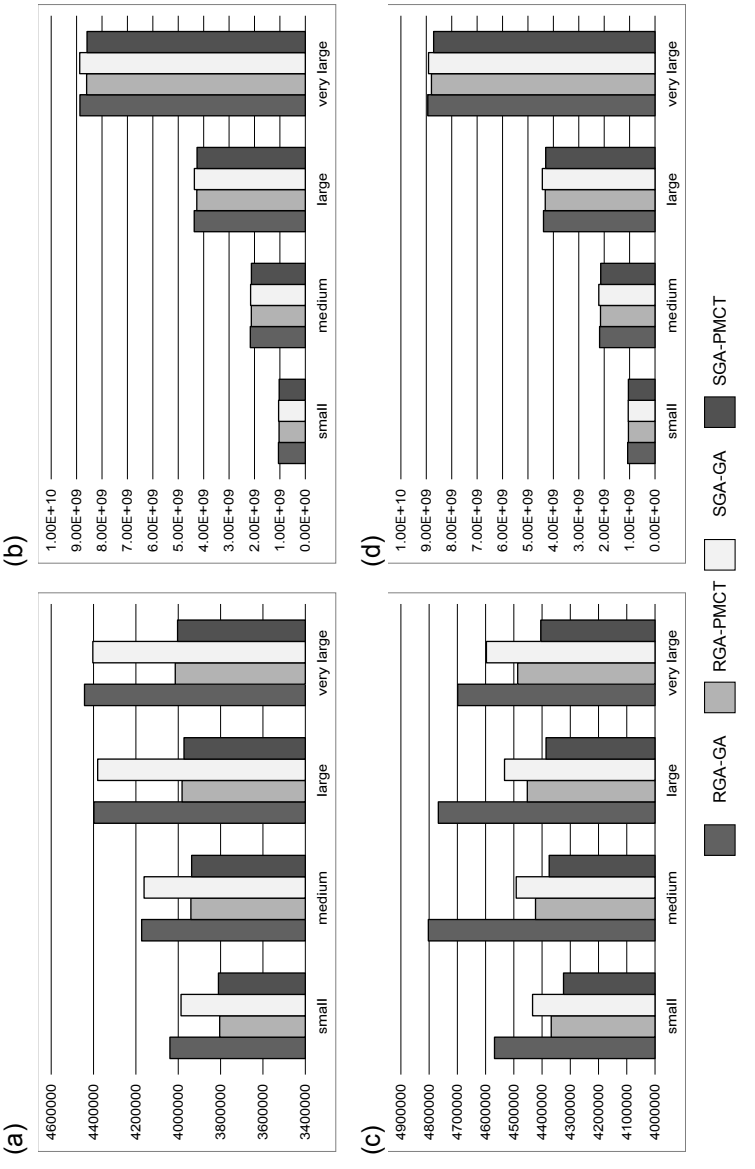


**Fig. 6.4** Experimental results for Stackelberg game: in static case - (a) average *Makespan*, (b) average *Mean_Flowtime* ; in dynamic case - (c) average *Makespan*, (d) average *Mean_Flowtime*.

In the symmetric game the best results for *Makespan* and *Mean_Flowtime* in all considered grid scenarios were achieved by *SGA-GA* scheduler. Especially in static 'Small' grid this method is very effective in *Makespan* reduction. The differences in the *Mean_Flowtime* results achieved by all hybrid meta-heuristics are not so significant, while in the case of *Makespan* both *SGA* hybrids significantly outperform **risky** hybrids in all grid scenarios.

In the case of Stackelbeg game two *PMCT* hybrids outperform the *RGA-GA* and *SGA-GA* algorithms. For *Makespan* values the differences in the results achieved by *PMCT* and *GA* hybrids are significant, while in the case of *Mean_Flowtime* all values are at the same level, except those obtained for 'Very Large' grid size. The best results in all instances are achieved by *SGA-PMCT* algorithm. However, in the case of static scheduling the efficiencies of *RGA-PMCT* and *SGA-PMCT* are very similar, while in the dynamic case, especially for *Makespan* values, the differences in both schedulers performances are significant.

The results achieved by two most efficient meta-heuristics in optimizing the users' game costs, namely $SGA-GA$ in the symmetric game and $SGA-PMCT$ in Stackelberg game, have been compared with the results generated by the best single-population security-aware scheduler from the previous chapter, namely $GA-SS-ANN$ algorithm. Tables 6.3, 6.4 and 6.5 present the comparison of the average values of $Maespan$, $Mean\_Flowtime$ and failure rate $Fail_r$ parameter (see Sec. 5.5.2 in Chapter 5).

It can be observed that both hybrid strategies outperform the $GA-SS-ANN$ algorithm in all but 3 cases. It confirms that game-based models are better adapted for the management of all security requirements in the grid system when compared to standard scheduling models, even if the res

Although the security requirements would imply some additional cost to the users of the grid system, it is worth assuming this cost in order to allocate tasks to trustful resources.

### *6.5.1 Computational Economy and Game-Based Models*

The experimental analysis presented in the previous section show that hybrid GA-based schedulers can be effective in solving the users games, however the main drawback of using such methods is their high computation complexity. The game scenarios presented in Sec. 6.3 are very general, which makes them useful in supporting the users decision process in various situations. In some real-life approaches the game scenarios are usually based on the well-known economical models.

Market-based approaches in grid computing enable grid resource owners, acting as sellers, to earn revenue by allowing others (mainly grid End-users, acting as buyers) to use their (idle) computational resources. The pricing of resources is driven by supply and demand. These models can be easily translated into the game-theoretical frameworks and are useful in grid resource management, as well as in defining users' decision strategies.

**Table 6.3** Average values of *Makespan* for $GA-SS-ANN$, $SGA-GA$ and $SGA-PMCT$ algorithms [$\pm s.d.$], ($s.d.$ = 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] |
| **SGA-GA** | **4104953.259 [± 379579.997]** | **4156536.877 [± 319105.946]** | 4264926.597 [± 548415.652] | **4353604.208 [± 595472.951]** |
| **SGA-PMCT** | 4185298.477 [± 574689.195] | 4162755.537 [± 444243.979] | **4260258.291 [± 676018.949]** | 4365824.522 [± 487088.573] |
| | | **Dynamic Instances** | | |
| **GA-SS-ANN** | 4141538.885 [± 247988.145] | 4212439.475 [± 342459.080] | 4232327.490 [± 333199.727] | 4364692.950 [± 339043.674] |
| **SGA-GA** | **4064399.586 [± 295082.511]** | **4159942.678 [± 573609.898]** | **4181202.361 [± 503195.319]** | 4330472.697 [± 485326.735] |
| **SGA-PMCT** | 4104009.894 [± 230614.767] | 4194715.259 [± 525510.365] | 4229959.495 [± 410287.752] | **4329168.378 [± 454255.168]** |

The following paragraphs present a general characteristics of the most popular economically- and game-based approaches for modelling users' relations and decisions in scheduling process.

### Commodity Market Model

This model is based on the Meta-broker architecture (described in Sec. 1.2.2). It is assumed here that the service providers primarily charge the end user for the resources they consume and the pricing policies are based on the demand from the users and the supply of resources. The resource owners and service providers are selfish in this approach and the end-users may or may not cooperate [26].

**Table 6.4** Average values of $Mean\_Flowtime$ for $GA-SS-ANN$, $SGA-GA$ and $SGA-PMCT$ algorithms [$\pm s.d.$], ($s.d.$ = standard deviation)

| Strategy | Small | Medium | Large | Very Large |
|---|---|---|---|---|
| | | **Static Instances** | | |
| **GA-SS-ANN** | 1098725220.445 [± 148984029.042] | 2261958805.835 [± 296213971.853] | 4395864089.470 [± 403819795.484] | **8705728350.062** [± 779128466.164] |
| **SGA-GA** | 1080025209.170 [± 106385883.899] | 2212272645.989 [± 225632106.035] | 7649954581.921 [± 564374456.205] | 8790927826,710 [± 820203622.476] |
| **SGA-PMCT** | **1039256248.489 [± 132828810.736]** | **2177583973.023 [± 279653260.144]** | **4251057955.321 [± 390873259.314]** | 8752787592.196 [±849851282.277 ] |
| | | **Dynamic Instances** | | |
| **GA-SS-ANN** | 1163342728.245 [± 136548966.434] | 2161846250.347 [± 272493690.708] | 4322245472.632 [± 533180226.552] | 8734534678.245 [± 835468708.749] |
| **SGA-GA** | **1124621170.786 [± 247484952.990]** | **2137984828.270 [± 143846367.588]** | **4254686510.766 [± 407990354.145]** | 8720343632.821 [± 931632311.801 ] |
| **SGA-PMCT** | 1155781873.098 [± 109523418.319] | 2200754844.400 [± 203859979.408] | 4293599602.333 [± 350634474.868] | **8712482470.785 [± 912872163.393]** |

### Auctions

In this model there are two groups of participants: sellers (resource owners) and buyers (grid end-users). The cooperation between users to form a coalition and win the auction is possible, but usually the users behave selfishly. The auction mechanism can be defined in many ways (e.g. English, Dutch, First and Second Price auctions). All of which differ in terms of whether they are performed as open or closed auctions and the offer price for the highest bidder. The users' strategies in particular auctions are discussed e.g. in [52].

### Bi-level Synchronized Auctions

The First Price bidding auction mechanism has been extended by Kwok et al. [92] to define the resource management and global scheduling policy at the intra- and inter-site levels in the 3-levels hierarchical grid structure. In the intra-site bidding each machine owner in the site, who acts selfishly, declares the "execution capabil-

**Table 6.5** Average values of failure rate $Fail_r$ parameter for $GA-SS-ANN$, $SGA-GA$ and $SGA-PMCT$ algorithms [$\pm s.d.$], ($s.d.$ = 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] |
| **SGA-GA** | 3.877% [± 0.98] | 4.356% [± 1.26] | 7.543% [± 1.89] | 9.015% [± 2.23] |
| **SGA-PMCT** | **3.738%** **[± 0.92]** | **4.005%** [± 1.05] | 7.456% **[± 1.35]** | 9.832% [± 1.56] |
| | | **Dynamic Instances** | | |
| **GA-SS-ANN** | 4.880% [± 0.98] | 6.097% [± 1.62] | 7.456% [± 1.32] | **7.026%** **[± 2.11]** |
| **SGA-GA** | 4.423% [± 0.73] | 5.533% [± 0.69] | 6.944% [± 0.98] | 7.046% [± 1.44] |
| **SGA-PMCT** | **3.875%** **[± 0.88]** | **4.542%** **[± 1.03]** | **5.953%** **[± 1.21]** | 7.211% [± 1.95] |

ity" of the resource. The local manager monitors these amounts and sends a single value to the global scheduler. In the inter-site bidding the global scheduler should allocate tasks according to the values sent by the local dispatchers. The authors prove that the cooperation of the players at both levels are the optimal strategies for both level-auctions. However, for the successful execution of all strategies some synchronization mechanism must be introduced, which can make the system in whole inefficient in a large-scale dynamic environment.

**Bargaining Models**

In this model the resource brokers bargain with resource providers for lower access price and longer usage duration. The negotiation process is guided by the end-users requirements (e.g., deadline) and can be provided directly between buyers (End-users) and sellers (resource owners). The most recent study on the bargain-

ing cooperative model application in optimizing the energy consumption in grid is proposed in [141].

## 6.6 Conclusions

This chapter showed the game-theoretic models as the effective methodologies for supporting the grid users' decisions, where the different scheduling criteria, including security and resource reliability, must be considered at the same time. The users' behavior can be effectively translated into the computational model linked to the grid scheduling. Due to large scale of the grid, the non-cooperative games seems to be a potential model for integrating various requirements in grid scheduling.

The users decisions in the scheduling process are modelled by using the two general non-cooperative game scenarios, namely symmetric non-zero sum game and Stackelberg game. The hierarchical procedure of solving those games is complex because of the need of integration and synchronization of two cooperating modules. However, the experimental analysis shows the high efficiency using the metaheuristics as the resolution methods for game-based models, especially in the case of additional security casts paid by the users. The game-based model concepts can be successfully implemented also in cloud computing, where the secure scheduling and information management remain challenging research problems.