J Intell Manuf (2015) 26:641–658
DOI 10.1007/s10845-013-0824-0

ALATO: An efficient intelligent algorithm for time optimization in an economic grid based on adaptive stochastic Petri net

Mohammad Shojafar · Zahra Pooranian · Mohammad Reza Meybodi · Mukesh Singhal

Received: 20 February 2013 / Accepted: 5 August 2013 / Published online: 27 August 2013

Abstract Cost and execution time are important issues in economic grids, which are widely used for parallel computing. This paper proposes ALATO, an intelligent algorithm based on learning automata and adaptive stochastic Petri nets (ASPNs) that optimizes the execution time for tasks in economic grids. ASPNs are based on learning automata that predict their next state based on current information and the previous state and use feedback from the environment to update their state. The environmental reactions are extremely helpful for teaching Petri nets in dynamic environments. We use SPNP software to model ASPNs and evaluate execution time and costs for 200 tasks with different parameters based on World Wide Grid standard resources. ALATO performs better than all other heuristic methods in reducing execution time for these tasks.

Keywords Grid computing · Petri nets · Learning automata · Optimization · Modeling

M. Shojafar (✉)
Department of Information Engineering, Electronics, and Telecommunications (DIET), Sapienza University di Roma, Rome, Italy
e-mail: shojafar@diet.uniroma1.it

Z. Pooranian
Department of Computer Engineering, Dezful Branch, Islamic Azad University, Dezful, Iran

M. R. Meybodi
Computer and IT department, Amirkabir Technical University, Tehran, Iran

M. Singhal
Electrical Engineering and Computer Science, University of California, Merced, CA, USA

Introduction

A Stochastic Petri Net (SPN) is a tool for studying systems. Useful information can be obtained by studying the behavior of dynamic systems modeled by SPNs, and the data can be evaluated to improve or change the system (Peterson 1981). SPNs are useful for mathematical and graphical modeling (Peterson 1981) and can be used to model, describe, and analyze the nature of concurrent, asynchronous, distributed, parallel, and uncertain or accidental systems (Reisig 1985; Jeng et al. 1999). In fact, Petri nets are component models that can concurrently show simultaneous actions and modes (Moore and Hahn 2003). The major difference between Petri nets and SPNs is that SPN tokens are divided into the usual indiscernible Petri net (where each node can handle each series of data or tokens with various input rates internally) and one distribution of each token with a unique feature; the relationship between general Petri nets and SPNs is similar to the relationship between high-level programming languages and assembly code. Finally, the graphical properties of SPNs (Hirel et al. 2000) allow systems to be studied through a visualization of the complexity of the simulation model. It is important to note that SPNs can be used to analyze a wide variety of systems, including grid computing networks (Venkataramana and Ranganathan 1999; Pooranian et al. 2013a; Pla et al. 2012).

One of the problems posed by SPNs is that they are not adaptable. SPNs have no access to previous information (Peterson 1981; Reisig 1985; Jeng et al. 1999; Moore and Hahn 2003). If more than one transition is enabled at a given time, each can be considered as the next step. This means that several events can occur simultaneously in SPNs and not as part of the same event. The events that occur do not change over time (Baranauskas et al. 2006; Zimmermann et al. 2001). This stands in contrast to the real world and its

dynamic nature. The main goal of a simulation model is to provide an evaluation of a system's performance through a simulation that is similar to the system. An SPN tool can identify problems and weaknesses in a system but is unable to solve problems or take actions to improve and optimize, and so the next state cannot be predicted (Schwardy 2001).

Grid computing, which in its simplest form is carefully distributed computing, has reached an advanced stage of evolution. A grid system is a simple but large and powerful virtual computer that is built from a vast number of computers and is capable of managing itself. It consists of a set of heterogeneous systems that are interconnected by a series of different complex combinations of shared resources. We cannot manage such complex systems with common approaches to resource management that try to optimize system performance generally. Rather, an economic approach is needed to provide a solid basis for the successful management of heterogeneous decentralized resources. Optimized resource allocation and scheduling is one of the vital factors in grid production environments that increases the grid system productivity (Arab et al. 2011). Several resource modeling have been proposed and applied in the recent years in several works (Pasandideh et al. 2011; Bilyk and Mönch 2010; Fleszar et al. 2011; Guinet 1995; Radakovič et.al 2011; Aissani et al. 2011; Archimede et al. 2013). Specifically, Archimede et al. (2013) introduces the architecture and behavior of (Distributed, Supervisor, Customer, Environment, and Producer) (DSCEP) framework under shared resources situation with disturbances. That uses a simple example of manufacturing system to illustrate the ability of DSCEP framework to solve the shared resources scheduling problem in complex systems such as Grid.

The most common economic model is the model of a commodity market, where resource prices are based on demand, supply, and value in the economic system. Grid users request a deadline (the time of program completion), budget, and time optimization strategy as quality of service (QOS) requirements. Economic scheduling in grid systems should use an efficient algorithm that allocates grid resources to user programs (which are composed of many independent tasks) so that the deadline and the optimization strategy are met in the minimum time, based on the desired parameters.

The economic approach assumes that grid owners (virtual organization that control a specific Grid) are looking for economic methods to manage their resources and schedule user requests to ensure that users receive the QOS they have requested. Competitive economic models provide algorithms, policies, and tools for sharing and allocating resources in an economic grid. We have developed an artificial intelligence (AI) method to decrease execution time when resources are allocated to independent tasks, based on a new modeling method (the adaptive stochastic Petri net, or ASPN). In previous work (Shojafar et.al 2011), an Adaptive Stochastic Petri Net (ASPN) is presented to improve the time in economical grid helping learning automata. We experienced large solve time variances, undermining the feasibility of this approach for more complex and large-scales scenarios. In the current paper, we extended our workload model to include applications that are associated with a heterogeneous data set, which allows us to take data transmission speeds and data locality into account during the scheduling process, and to support the online arrival of applications.

The rest of this paper is organized as follows. Section "Related work" describes related work, including learning automata (LA), on which the new ASPN method is based. Section "The system model" introduces the concept of being adaptive and presents our new ASPN model, and Section "Our proposed approach (ALATO)" proposes a resource allocation method, called "ALATO". Section "Simulation and efficiency evaluation" compares the performance of ALATO with previous methods in different scenarios and presents the results with graphs and diagrams. Section "Conclusion" presents our conclusions and future research directions.

Related work

Methods exist for introducing adaptive characteristics in large Petri net models (Murata 2002; Zhou and Jeng 2002). These methods use AI techniques such as artificial neural networks, fuzzy logic, knowledge-based systems, and stochastic LAs. They attempt to develop techniques that can significantly improve the QOS in Petri nets. This research falls into two general categories (Zhou and Jeng 2002). Research in the first category, fusion hybrid, uses AI to model real system executions with the aim of defining the conditions under which adaptive Petri nets can capture the information that is available in real systems. These models provide a view of the real world that can easily be analyzed, processed, and evaluated to produce results that can be applied in the real world. Adaptive ability is concerned with making fundamental changes in a Petri net configuration to match it to a real system. The structures of Petri nets have been changed in this group (first category) to make use of AI techniques that have the ability to learn and thus increase the power of the Petri nets. However, hybrid fusion Petri nets have some disadvantages. For example, the learning that takes place often increases computational complexity and overhead, so that a large amount of time is needed to model all the characteristics of real systems. Petri nets in the second category are combination hybrid Petri nets. A relatively small number of researchers work in this area to implement intelligent adaptive techniques along with Petri net methodologies for real-world problems. The combination hybrid models have less complex designs than the hybrid fusion models and gen-

erally require less execution time. Combination hybrid models combine different technologies with modifications that enable the models to run in environments with greater efficiency and accuracy than they normally would. These models use the basic properties of Petri nets and apply AI techniques to the internal task demands and to every part of the real system.

A real-time mixture of intelligent features has been proposed in Khosla and Dillon (2002), Jain and Martin (1998) to improve current systems: the ASPN, a stochastic Petri net that can predict the next state of the system based on information about the environment achieved in the current state of the system and gradually adjusts to environmental conditions. ASPNs can be used in intelligent systems and dynamic systems that undergo large changes. The ASPN algorithms presented in this paper belong to the hybrid fusion category.

Several time-optimization algorithms have been proposed for resource allocation and scheduling in economic grids. The goal of all these algorithms is to make the best use of the users' budgets and possibly reduce the execution time. The Buyya Time Optimization (BTO) algorithm presented in Buyya (2002), Al-Ali et al. (2002) aims to optimize execution time and is evaluated in Czajkowski et al. (2002). BTO is a heuristic algorithm that answers to a considerable extent the call to provide independent homogeneous or acceptably heterogeneous tasks for the user. BTO assumes that resources are time shared, and this assumption is responsible for some complexities and shortcomings of the algorithm. Consequently, Mahdavifar and Meybodi (2007a, b) presented different methods to optimize and simulate BTO, including the Extended Buyya Time Optimization (EBTO), Advanced Extended Buyya Time Optimization (AEBTO), and Learning Algorithm Time Optimization (LATO) algorithms. Despite current research efforts (Shojafar et al. 2013; Pooranian et al. 2012, 2013b, c), the cost-optimal and time-optimal resource scheduling and allocation of external resources while taking into account the availability of a local IT infrastructure remains an open problem.

BTO, EBTO, and AEBTO are heuristic algorithms, and LATO is an AI algorithm based on LAs. AEBTO changes the structure of BTO to provide better scheduling results, but all three heuristic methods use time-shared allocation and are not appropriate for optimizing task completion time. None of the heuristic methods use AI algorithms, so their optimizations involve several additional loops that are not essential for resource allocation. LATO uses LAs to minimize the completion time within the budget and utilizes space-shared allocation. Other methods for resource allocation using an economic grid are presented in Reddy (2006), Assuncao and Buyya (2006), Mirzaee and Rahimzadeh (2011), and a framework for an economic grid scheduler is presented in Al-Khasawneh and Bsoul (2010). Although this framework has a good structure, it is not suitable for discrete high performance requests. Finally, a new method based on LAs was recently proposed in Sarhadi and Meybodi (2010), but it focuses on optimizing the budget rather than on modeling patterns to improve the system performance.

The system model

AI algorithms are currently applied in SPNs to decrease the time needed for distributing tasks and to simulate real systems, modeling various features and monitoring their status to resolve any problems that may exist. One of these algorithms is the stochastic LA (Boppana and Halldorsson 1992; Bui and Eppley 1995), which can be considered to be single object that executes several actions. An LA's performance at any time involves selecting an action from a set of actions and evaluating the action in a random environment, using the response from the environment to select its next action. Hence, an LA gradually identifies optimal actions. The method that an LA uses to select the next action is deterministic. LAs can have fixed or variable structures. A variable LA is a set consisting of quadruples{α, β, p, T}, where α is an action, β is an input, p is a probability for the action, and T is learning algorithm. LA algorithms can be classified as S and P models. In a P model, or a standard model, the value of a reward (when a desirable response is received) or penalty (when a non-optimal or non-desirable response is received) is constant and is the same for each action. The general form of a standard-model LA is given by Eqs. 1 and 2. Equation 1 applies when a desirable response is received from the environment, while Eq. 2 applies when a non-desirable response is received. In Eq. 1, the probability of the action is increased by the reward rate while the probabilities of all other actions decrease. In Eq. 2, the probability of the action is decreased by the punishment rate while the probabilities of all other actions increase. The parameter r indicates that the system has at most r actions and a is the reward parameter and b is the punishment parameter.

$$\begin{aligned} \mathrm{p_i}(\mathrm{n}+1) &= \mathrm{p_i}(\mathrm{n}) + \mathrm{a}[1 - \mathrm{p_i}(\mathrm{n})] \\ \mathrm{p_j}(\mathrm{n}+1) &= (1-\mathrm{a})\mathrm{p_j}(\mathrm{n});\ \forall \mathrm{j};\ \mathrm{j} \neq \mathrm{i} \end{aligned} \tag{1}$$

$$\begin{aligned} \mathrm{p_i}(\mathrm{n}+1) &= (1-\mathrm{b})\mathrm{p_i}(\mathrm{n}) \\ \mathrm{p_j}(\mathrm{n}+1) &= \frac{\mathrm{b}}{\mathrm{r}-1}(1-\mathrm{b})\mathrm{p_j}(\mathrm{n});\ \forall \mathrm{j};\ \mathrm{j} \neq \mathrm{i} \end{aligned} \tag{2}$$

In an S model, the response of the environment, defined as a function of the automaton, is added to the standard model to improve the automaton's learning. There are three types of S models, based on the reward and punishment parameters a and b: when a = b the S model is called $S\text{-}L_{RP}$, when $b << a$ the S model is called $S\text{-}L_{ReP}$, and when $b = 0$ the S model is called $S\text{-}L_{RI}$ (Narendra and Thathachar 1989; Poznyak and Najim 1997). In this paper, we have used all three S models of

automata by the names of S-L_{RP}, S-L_{RI}, and S-L_{ReP}, respectively. Learning automata update the probability vectors for r actions. If action α_i is selected in the nth iteration and the environment's response is $\beta_i(n)$, then the automaton's probability vector is updated as in Eqs. (3) and (4):

$$p_i(n+1) = p_i(n) + a(1-\beta_i(n)) \times(1-p_i(n)) - b\beta_i(n)p_i(n) \quad (3)$$

$$p_j(n+1) = p_j(n) - a(1-\beta_i(n))p_j(n) + b\beta_i(n)\left[\frac{1}{r-1} - p_j(n)\right] \forall j \quad j \neq i \quad (4)$$

We now introduce two additional concepts that are used in this study: *time-shared* resources and *space-shared* resources, which play a crucial role in task scheduling in the economic grid. A task that can be assigned to a time-shared resource is immediately transferred to run on one of the CPUs, because time-shared CPUs can simultaneously execute multiple tasks. In this case, each task is allocated a specified time interval to execute on the CPU. Then it is suspended and the next task is transferred to run on the CPU. When a task finishes, it is removed from the execution queue for the resource, and the CPU time is divided equally among the remaining tasks. One issue that arises with time-shared resources is that we cannot predict the completion times for tasks that have been assigned to them, because of the possibility of assigning new tasks to these resources. Another problem is that scheduling several tasks may overload the CPU. This happens when a large number of simultaneous tasks run on a time-shared CPU, so that only a very small percentage of CPU time is allocated to each task and there is frequent suspension of each task's execution to resume the execution of another task—which incurs CPU overhead and thereby reduces efficiency.

In contrast, with a space-shared resource, the tasks in the queue are not processed as soon as the CPU becomes idle. A space-shared CPU can or wants to run only one task. This means that when a task is transferred to a space-shared CPU for execution, it continues to execute until it completes. Therefore, we called the CPU a non-preemptive resource in this case. In this paper, we use space-shared resources for independent tasks, based on AI algorithms. Space-shared resources are divided into two scheduling classes: *all-at-once allocation* and *stage allocation*.

In stage allocation, the scheduler schedules in stages that continue until all of a user's tasks have been assigned. Each resource receives a number of tasks equal to the number of its active processors, which prevents concurrent task execution on one processor. A queue is also formed for each user and unassigned user tasks are stored in the queue after the set of independent tasks that constitute the user's application have been created. In the next stage, the rescheduling algorithm executes: resources are selected for mapping and tasks that have not run can be delegated to the available processors. This procedure continues until all tasks have been assigned to resources.

In all-at-once allocation, the scheduling algorithm assigns each task once to each resource, which controls and manages the number of tasks running on each of its processors (no processor runs more than one task simultaneously). Because it has task queues for each resource, the scheduling algorithm will pass these tasks to unemployed CPUs.

The difference between the two types of scheduling is that in stage allocation scheduling, tasks are stored in their users' queues and are assigned to resources simultaneously and gradually in a series of repeated steps, while in all-at-once allocation scheduling, all tasks are assigned to resources once and the gradual transfer of tasks to processors is performed for each resource (each resource has a queue of its assigned tasks), which means that resources are treated the same as space-shared resources.

This paper proposes grid architecture based on SPN modeling in an adaptive form. The grid architecture consists of three layers. The top layer is an application layer with a task flow environment in which users describe their applications. The bottom layer is a fabric layer that consists of grid nodes connected in a net. Between these two layers is a middleware layer that is responsible for resource allocation. Figure 1 shows the three layers of the grid net in an SPN that is implemented in SPNP software. Each request is sent from the application layer to the middleware layer, and after the request is analyzed into independent tasks and subtasks, these are sent to the fabric layer that controls hardware and grid resources. Figure 2 shows the task allocation procedure in the middleware layer.

In Fig. 2, 200 ordered tasks are given to the scheduler. The scheduler considers the tasks to be the different actions of an LA. Each request is converted to a queue of requests that are referred to different resources.

In the middleware layer, user programs are received from the application layer, and after dynamic information about resources has been obtained, task scheduling requests are given to the scheduler module. Our proposed algorithm sorts tasks in descending order of execution time, as shown in Fig. 3. Initially, a general model of the grid system with 200 tasks is considered. The variable *genC* is the loop control variable for the loop that sorts the tasks. The terminating condition is given in the *CaseIDCounter* state. In the second loop, orders are sorted in descending order of execution times.

Each token represents a user's task; it includes information such as the task length, a priority number based on the task length, and an LA. As noted, our proposed SPN falls in the fusion hybrid category. This category has some disadvantages. Because Petri nets and AI techniques are combined, learning in fusion hybrid Petri nets often results in compu-

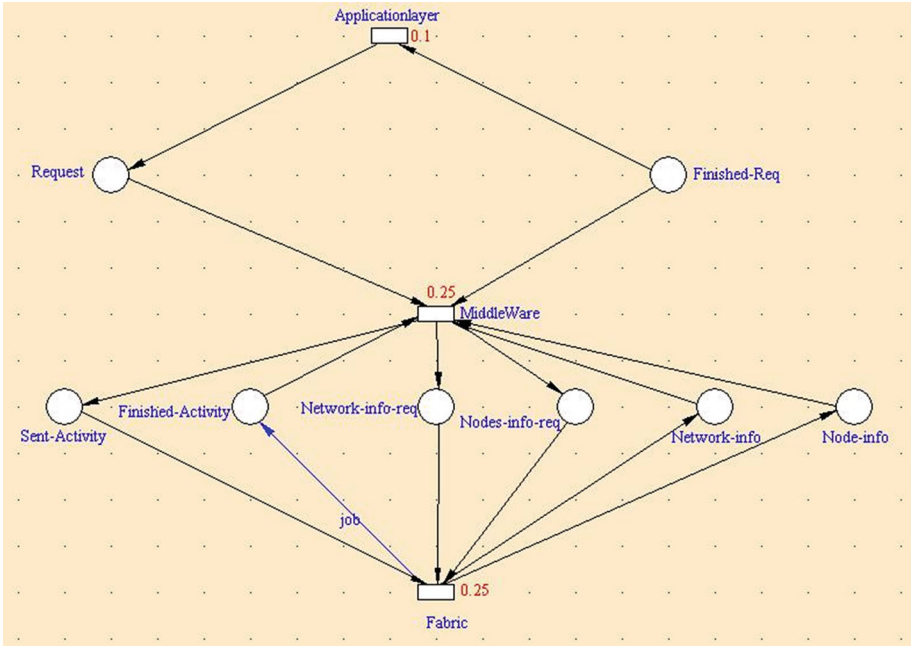

Fig. 1 The three layers of the grid system

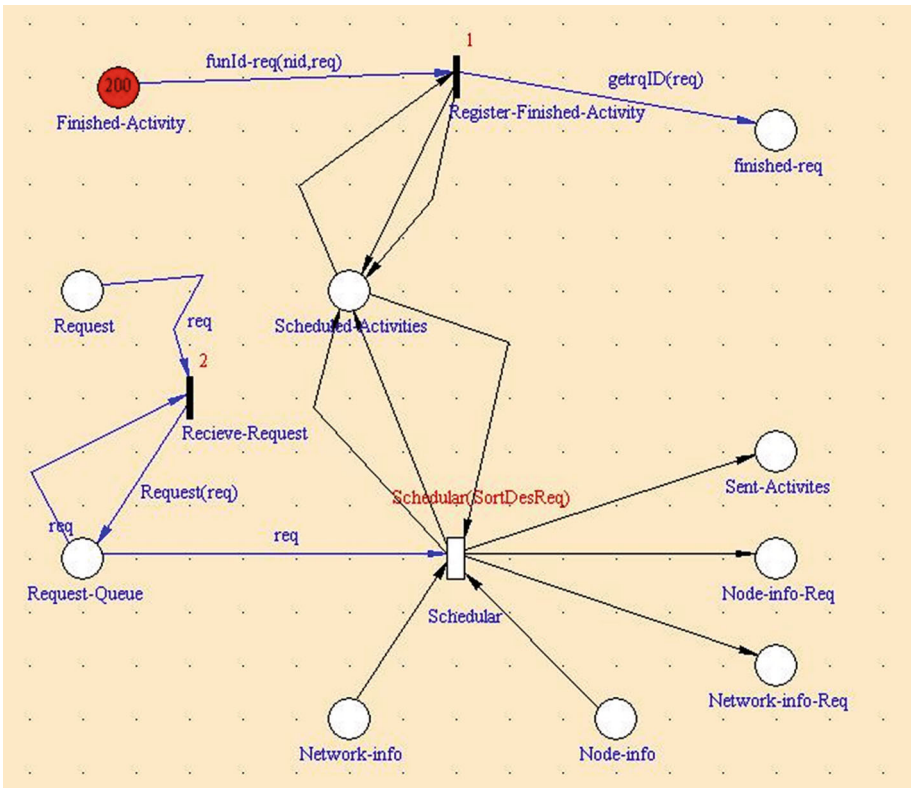

Fig. 2 Task allocation in the middleware layer

tational complexity and overload. In the ASPNs in this category, AI techniques such as artificial neural nets, fuzzy logic, and knowledge-based systems are used.

However, the ASPN that we propose is based on LAs. Unlike ASPNs that use artificial neural nets, ASPNs based on LAs do not need to be trained with a series of preexisting patterns. Therefore, our proposed ASPN is quite flexible for systems with continually changing and relatively unknown environments, where there are no preexisting patterns for training artificial neural nets. Hence, our proposed ASPN does not share the problem of computational complexity and overload that other fusion hybrid SPNs have.

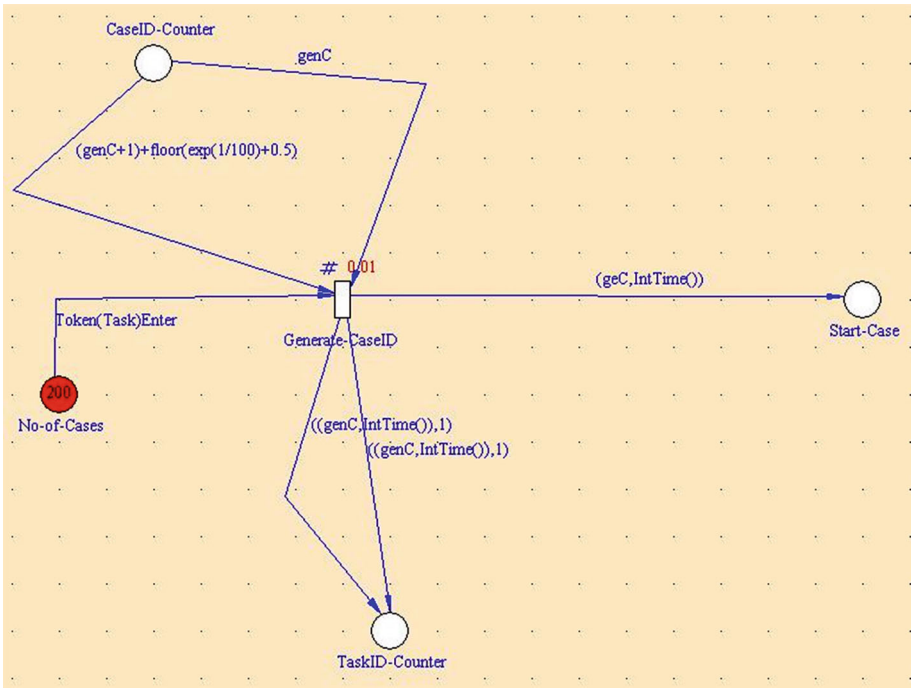

Fig. 3 Sorting tasks in descending order

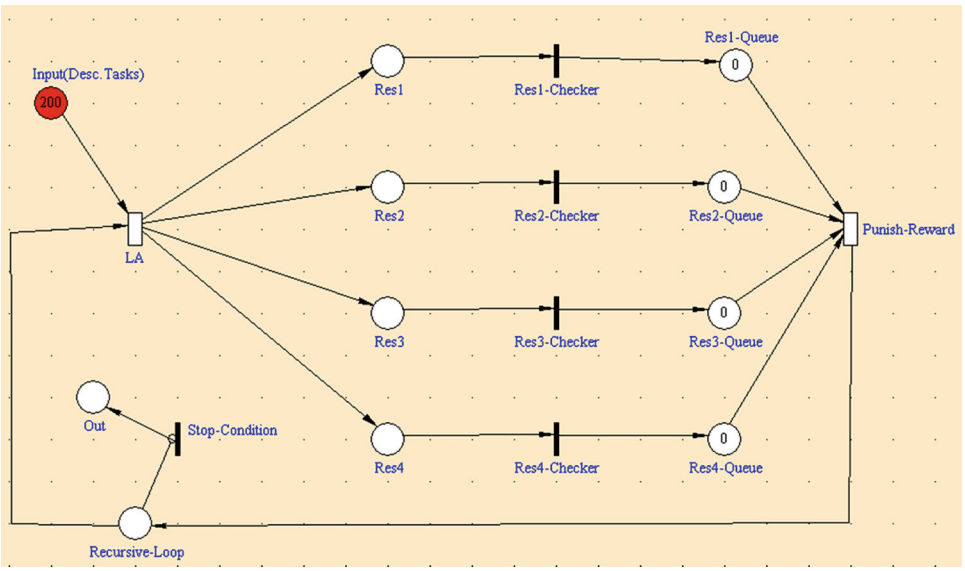

Fig. 4 Assigning tasks to resources and punishments and rewards for 1,000 loop iterations

In addition, in LA-based ASPNs, every token has its own automaton and, unlike previous adaptive models that require creating new places and transitions for developing the artificial neural net, there is no need for new places and transitions. Therefore, no complexity is involved in the design and there are no increased calculation overloads.

In Fig. 4, every token selects an operation (one of the resources in the grid) according to its probability vectors. Then all tasks assigned to each resource are evaluated according to the resource's capacity and the deadline that the user has specified for completion of the task, and each operation is punished or rewarded. *Res1*,*Res2*,*Res3*, and *Res4* are the four candidate resources for assignment. Each of these has a separate queue: *Res1_Queue*, *Res2_Queue*, *Res3_Queue*, and *Res4_Queue*, respectively. In assigning tasks to resources, the numbers of the tasks are placed in these queues in ascending order of execution time, and the *Punish-Reward* transition assigns a punishment or reward to each resource. This routine is recursively performed 1,000 times on the resources, with the result of each iteration saved in *Recursive_Loop* before proceeding to the next stage.

Fig. 5 Scheduling algorithm

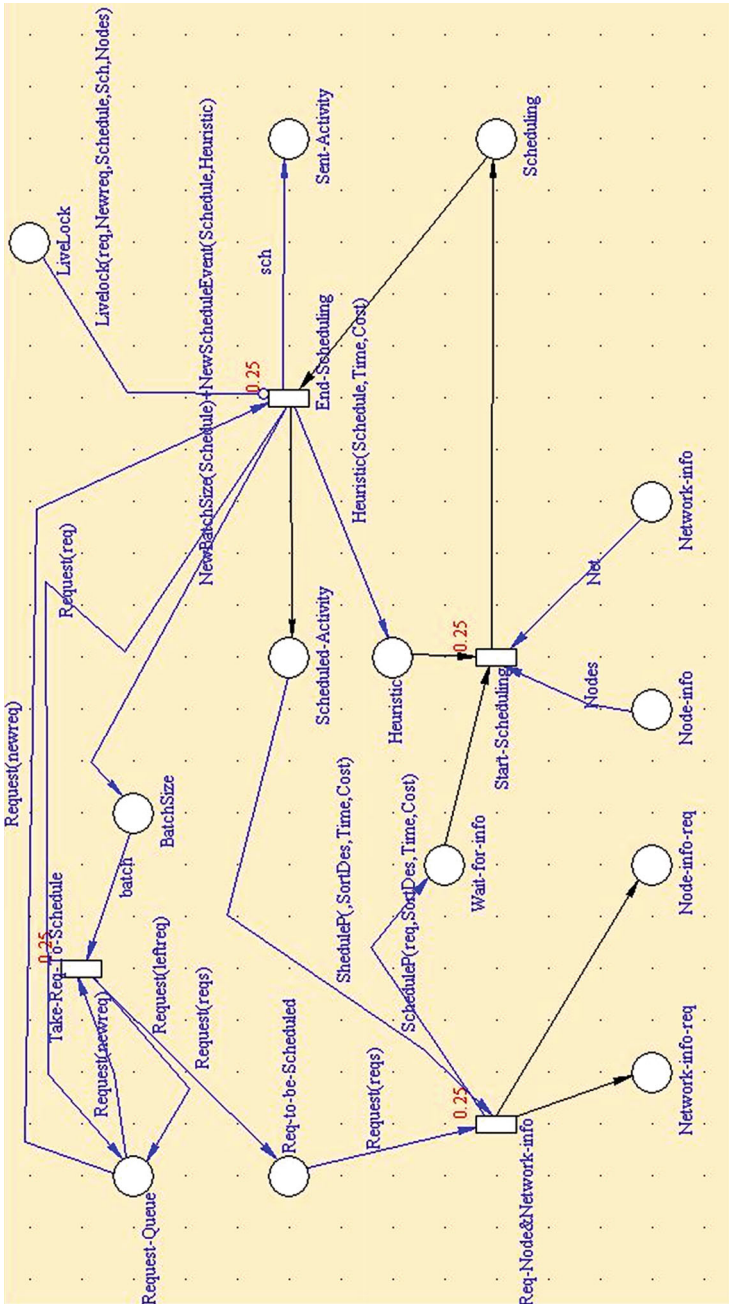

Figure 5 shows the scheduling and assignment algorithm for resource requests.

The tasks belong to applications that users have sent to the grid for execution. Each user determines the QOS parameters for execution of the application: the deadline, the budget, and the optimization strategy to be used by the system. Simulated algorithms can use time or budget optimization strategies, or both. A user who chooses the time optimization strategy for scheduling expects that the grid system will complete execution of the user's application within the determined budget and will minimize its execution time as much as possible. Through the simulation, it can be determined which of the time optimization algorithms will yield the shortest application execution time, as the efficiencies of the time optimization algorithms are compared.

Grid environment

The simulated environment for the grid consists of one or more users and a number of resources. Users may enter the grid system at any time and submit their applications for execution. An application consists of independent tasks that can each be executed on a desired resource. After entering the system, the user acquires a mediator that performs its

application scheduling, and after the application's tasks have been assigned to resources and have completed, the results are given to the user.

User model

In the simulation experiments, the simulated environment is a single-user environment unless a multi-user environment is being modeled, and algorithms can be compared under this condition. A single-user model does not mean that only one user enters the system during the simulation, but rather that there will be no interference between the scheduling of distinct users. We cannot use two separate simulations to compare algorithms because the applications should be homogeneous.

When entering the system, users specify their budgets and deadlines. Different experiments use different values for these. We assume that submitted applications can always be executed within the specified budget and deadline. If a user requests cost optimization, then after examining the execution possibilities, the scheduling system should complete the application within the time deadline and at the least possible cost. For example, if a user specifies 200,000 for the budget and 10,000 for the deadline and enters the system at time 5,000, the system should deliver the results of the user application by time 15,000 and with the least possible cost (which must, of course, be less than the budget).

Application model

An application consists of several independent tasks. Our experiments always use 200 tasks. Each task has its own length, expressed in millions of instructions (MI). Task lengths are assumed to be variable. A (*minimum*… *maximum*) range is selected for the task lengths and a value is randomly selected from this range for each task length, so that the task length distribution is uniform. The most homogeneous task length distribution range is (100,000 … 110,000), and the most heterogeneous is (10,000 … 200,000). The mean range is always set to 105,000, so that it is possible to compare an algorithm with different task length heterogeneities. For convenience, the length unit is sometimes set to 1,000 MI (Million Instructions), so that in our examples we show the most heterogeneous state range to be (10 … 200).

Also, the tasks we study are computation intensive, and they obtain their necessary values from an input file at the outset and place their results in an output file at the end. The tasks' short interactions with their data are neglected, so that only their computational aspects are considered.

Our implementation defines the task model in SPNP software's CSPL language http://www.cslibrary.org/research/language.html.

Resource model

We use a sample construction named "GR1" for the grid resources in our experiments. Although our construction is a simulation, it is based on a real construction, GR1, that is described in Buyya (2002). The properties of resources in GR1 are such that different schedules can be meaningfully compared. All of these resources have a processor and are used as shared space. Table 1 shows these constructions. In this table, the resources are sorted in ascending order of cost and are named accordingly, so that the workload of each resource and its assigned tasks can be examined more easily. For resource sorting, the order of the effective costs G$/MI that are used is the same as the order of the declared costs G$/sec. This is the desired order for cost optimization algorithms. The algorithms begin with the least expensive resource, *R1*, for task assignments.

As the declared costs of resources increase, their computational power (instructions executed per time unit) increases. The last column displays the effective cost, which combines the cost and power of each resource. The effective cost is the declared cost divided by 1,000 times the resource power. It should be noted that by cost (expensiveness or inexpensiveness), we mean effective cost (G$/MI), and that the declared cost (G$/sec) is of no use because no results can be derived from it.

The code for GR1 and its resources is shown in Fig. 6.

If only the declared cost of resources were considered, the conclusion would be that resources with greater power cost much more than resources with less power and that their use is too expensive for users. However, it should be noted that it seems that using *R4* instead of *R1* would not be cost effective, because the power of a resource should be considered along with its cost. For example, if the declared cost for a resource is twice the cost of another resource, but its power is also twice as great, the cost optimization algorithm will consider the two resources to have the same value.

Common stages of the optimization algorithms

The optimization algorithms perform some common actions before scheduling. These actions can be classified into the following three stages:

Table 1 GR1 construction for resources

Effective cost (G$/1,000MI)	Declared cost (G$/sec)	Implementation rate (MI/sec)	Resource name
5	0.5	100	R1
6.25	1.5	240	R2
8.33	2.5	300	R3
12	6	500	R4

```
void   Res(GR1   R[4])//Resource   policy
(FIX)
{      R[1].benf_cost=5;
       R[1].cost=0.5;
       R[1].Run_Rate=100;
       R[1].no=1;
       R[2].benf_cost=6.25;
       R[2].cost=1.5;
       R[2].Run_Rate=240;
       R[2].no=2;
       R[3].benf_cost=8.33;
       R[3].cost=2.5;
       R[3].Run_Rate=300;
       R[3].no=3;
       R[4].benf_cost=12;
       R[4].cost=6;
       R[4].Run_Rate=500;
       R[4].no=4;
}//GR1 Difinition finished
```

Fig. 6 GR1 code and resources

- *Resource identification*: Resources that can be used to execute tasks are detected, and their power and properties are determined through the grid information service (GIS).
- *Resource trading*: Each resource is defined in terms of cost per time unit (cost in seconds: G$/sec), then in terms of the resource power per time unit (at the rate of one million instructions per second: MI/sec), and the real effective cost of the resource is then determined—this is the cost of executing one million instructions (cost per million instructions: G$/MI).
- *Resource sorting*: Resources are sorted in ascending order according to their effective costs (as determined in the previous stage). Resources with lower costs are preferred in task assignment. If two resources have the same effective cost, the resource with more power is preferred so that the algorithm has the greatest possible efficiency. Thus, in order to produce the best ordering, resources are sorted by two parameters: *effective cost* and *power*.

In the next section we explain ALATO, our proposed LA-based algorithm.

Our proposed approach (ALATO)

In what follows, the *Minimum* algorithm is the algorithm with the shortest execution time. To attain the execution time of the Minimum algorithm, we compare algorithms for time optimization; rather than scheduling tasks, scheduling aims for the shortest execution times. We assign a budget to each million-instruction unit and consider budget when selecting the appropriate resource for execution. Another way to calculate computation time is to use our proposed algorithm to obtain the minimum cost. The minimum cost method is implemented by considering completion deadlines and studying the resultant costs. A certain amount of time is then subtracted from the deadline and the minimum cost method is re-executed. This is done until a deadline is obtained in which the minimum cost of the task execution is almost the same as the budget specified by the user. In this case, the deadline used in the minimum cost method is the same as the minimum computation time.

The ALATO algorithm that we propose in this section is *space-shared* and uses *all-at-once* scheduling. It sorts tasks into queues in descending order according to their execution length, which reduces the idle time of the more expensive resources. A user who enters the system is accepted if the user's application can be executed within the specified deadline and budget, and once a user is accepted, completion of the user's application satisfying these ranges is guaranteed. For this purpose, at the end of the ALATO algorithm, after scheduling tasks (mapping tasks to resources), a simple acceptance control stage studies the total time of task execution and the cost of the application execution. If a user is accepted, a distribution stage is executed in which the user's tasks are assigned to the resources specified in the scheduling.

The ALATO algorithm makes some changes to the implementation and the system of rewards and punishments in the LATO algorithm, and it yields the best results of all the algorithms mentioned. ALATO attempts to perform a complex time optimization by assigning LAs to tasks, which is more effective than the way LAs are used in exploratory algorithms. The LAs have variable structures and their actions are grid resources. The LA assigned to a task selects the resource to which the task should be assigned. When all tasks have selected their desired resources, they are rewarded or punished by the environment. Then all tasks perform another resource selection that produces a new environment response. This is repeated an indefinite number of times until all tasks have found their appropriate resources and are assigned to them.

Thus, the ALATO algorithm does not reach its final scheduling through a single identification of tasks and resources: rather, it performs several iterations, with a new schedule produced in each iteration. The environment responds to tasks (punishes or rewards them) according to the resources they select. The tasks use this response in the next iteration to produce a new schedule that is closer to the optimum schedule. These iterations are repeatedly performed until all tasks find their appropriate resources and no longer change their decisions. That is, the iterations continue until all LAs converge and produce the same schedule in subsequent iterations. This is the final schedule that will assign tasks to resources.

In each iteration of the ALATO algorithm, the schedule that is produced is temporarily applied to the resources so that it can be evaluated. For this purpose, every resource has a temporary queue called the "assignment queue" that is empty at the beginning of each iteration and to which tasks are added when they select that resource.

The environment's responses to the task LAs are determined by a time optimization algorithm that considers ideal conditions. The environment uses the following three criteria for rewarding or punishing tasks:

1- The total cost of the task execution should not be greater than the budget specified by the user.
2- The execution time for tasks assigned to the busiest resource shouldn't be very different than for other resources.
3- Each task should be assigned to a resource that can complete its execution earlier than other resources.

The environment punishes tasks on the basis of the first two criteria and rewards them according to the third. Concerning the first criterion, if the total cost of task execution for the selected resource is greater than the specified budget, the task is punished. Punishment begins from the end of the task queue for the most expensive resource to which at least one task has been assigned. To punish these tasks and eliminate them from the selected resource's assignment queue, the difference between the cost of their execution using this resource and their execution using the least expensive resource is subtracted from their total execution cost. Punishment of a task continues until its total cost is less than the specified budget. The effect of this punishment is that tasks are directed to less expensive resources, and the purpose of this stage is to quickly lower costs and bring them to a value within the specified deadline by transferring tasks from the most expensive to the least expensive resource. On the basis of the second criterion, the environment punishes tasks in the assignment queue for the busiest resource.

It should be noted that the busiest resource is not necessarily the most expensive resource. We begin by selecting Y tasks from the end of the busiest resource's queue. If another resource used in the scheduling (with at least one assigned task) other than the most expensive resource can process a task more quickly than the busiest resource, the environment punishes the task so that it will select a more appropriate resource in the next iterations; otherwise the task is not punished.

This is done for all Y tasks in the busiest resource's queue and leads to a more balanced distribution of resources for the tasks. The value Y is calculated by Eqs. (5) and (6) and considering Table 2 as follows:

$$X = (\text{Taskno} - A)/(\text{resourceno} - 1) \tag{5}$$

Table 2 Summary of notations

Item	Marks
Task numbers	Tasksno
Resource number	resourceno
Numbers of task are assigned to the most expensive resource	A
Numbers of tasks are assigned to the most prolific resource	B

where A is the number of tasks that are assigned to the most expensive resource, $resourceno$ is the number of resources, and $Taskno$ is the number of tasks. X is thus the mean number of assigned tasks for *resourceno-1* resources; it is the number of tasks that should be distributed to each resource excluding the most expensive one. (The reassignment of tasks to resources excludes the most expensive resource because it increases execution costs.)

$$Y = (B - X) * \frac{\text{resourceno} - 2}{\text{resourceno} - 1} \tag{6}$$

where B *is the number of tasks that are assigned to the most prolific resource*. Y is then the number of tasks that will be selected from the end of the busiest resource queue. B is always equal to or greater than A, because the number of tasks in the busiest resource queue is greater than X, the mean number of tasks distributed to other resources. The value (*resourceno*-2) / (*resourceno*-1) is the ratio of the mean value of the executable resources, excluding the most expensive resource and the busiest resource, to available resources other than the most expensive one.

Eventually, the environment rewards tasks according to the third criterion. Tasks that have been able to attain their completion time during or earlier than the previous iteration are rewarded, provided that they have not been punished in this iteration. This encourages every task to select a resource that can execute it in the shortest time.

The ALATO algorithm initially sorts tasks in descending order according to their execution lengths. It then finds the appropriate resource for each task's execution in a finite number of iterations. This algorithm eventually assigns tasks to those resources, using all-at-once allocation for the scheduling. In order to attain the best solution, various models of standard LAs and various S models were examined. In standard automata, the environment uses two punishment responses and one reward response during the algorithm's learning, because if a task has not been punished according to either of the first two criteria mentioned above and it has reached its completion time during or before the previous iteration, it has possibly found an appropriate resource for its execution. The punishment value in this state (0.1) is greater than in the first state (0.01) and the second state

Table 3 Conditions for the time optimization test with fixed budget

Parameter	Value	Parameter	Value
Algorithm	Variable	Deadline	50,000
User number	1	Budget	150,000
Task no.	200	Task length range	[10 . . . 200]
Resource configure	GR1	Test no.	20

(0.05). The punishment value in the second state is greater than in the first because shorter tasks have been assigned to the busiest resource and the task accumulation for this resource has increased. The increased punishment causes tasks to place themselves in queues for less busy resources and thus increase their load balance in the next iteration, and it also significantly increases the rates at which the LAs converge. S-model LAs were tested with different punishment and reward values. These tests are discussed in the next section, which evaluates BTO, AEBTO, LATO, the Minimum algorithm, and our proposed ALATO algorithm for three different LA states.

Simulation and efficiency evaluation

For our implementation, we defined a task model in SPNP software's CSPL language (Buyya 2002). Four resources with different execution rates and effective costs were used and were sorted in ascending order of cost. In our simulation, 200 tasks of variable lengths with times specified by the user were examined, with different budget heterogeneities. In our proposed ASPN, every token represents one user task and includes information such as task length, priority number based on task length, and the LA.

We tested our proposed algorithm in the following three cases. Note that in the mentioned cases, ALATO performs better than all related methods, specifically LATO method.

Time optimization within specified time and budget (case I)

In the first case, we tested our proposed algorithm with various LA models under the conditions listed in Table 3. In order to make appropriate comparisons, the task lengths were considered to be the same in all models, with a uniform distribution of task lengths within the range specified by the task length parameter. To obtain the execution times under the available algorithms for a specified deadline, 20 tests were performed and the mean value was chosen for examination.

The conditions were assessed in the different states of the standard automata (P-L_{RP}) and *S*-model automata (S-L_{RP}, S-L_{RI} and S-L_{ReP}).

Table 4 Changes of punishment and reward rate in the P-L_{RP} model and in time optimization algorithms with a fixed budget of 150000

(a,b)	State 1	State 2	State 3	State 4
Algorithms	(0.01, 0.01)	(0.05, 0.05)	(0.1, 0.1)	(0.5, 0.5)
BTO	17,440	17,900	17,950	18,543
AEBTO	17,240	17,440	17,645	17,840
LATO	17,240	17,314	17,420	17,580
ALATO	17,175	17,275	17,354	17,408
Minimum	17,150	17,150	17,150	17,150

P-L_{RP} model

For this model, the results for different punishment and reward rates were compared. In this (P) model, response $B_i(n)$ with probability C_i is 1 (desired response) and with probability 1-C_i is 0 (undesired response). Because two algorithms based on learning automata, LATO and ALATO, were implemented, so the punishment and reward rates for a budget of 150,000 were considered for these algorithms. The results for the standard automata are shown in Table 4. Since the punishment and reward rates are equal, they should be small to enable the algorithm to gradually reach the appropriate schedule. For a fixed budget of 150,000 with the same punishment and reward rates, 0.01 is the best reward/punishment rate for the LA-based algorithms.

Since the ALATO algorithm considers more task states, ALATO's LAs learn faster than LATO's, as shown in Fig. 7. By increasing the values of a and b, the growth of ALATO's execution time gradually becomes slower than in LATO.

In Fig. 7, the variable *Delta_ALATO* represents the time difference between the ALATO algorithm and the Minimum algorithm with different values for the rewards/punishments, and *Delta_LATO* represents the same for the LATO algorithm. The linear *Delta_ALATO* slope is less steep than the linear *Delta_ALATO* slope, which means that ALATO learns more quickly than LATO. In particular, ALATO assigns tasks to resources more quickly than LATO. Also, setting a higher reward and punishment rates just make a worse schedule, so, we try to decrease these rates with respect to their run-time.

S-L_{RP} model

In *S*- model methods, if operation αi is selected in the nth iteration, the undesired environment response to this selection is $\beta i(n) = 1$ and the desired environment response is $\beta i(n) = 1/(1 + previous\,time/new\,time)$—i.e., when a less expensive resource is selected, a better response is received from the environment. According to the results presented in the simulation section, the S-L_{RP} method yields the best result with a reward rate of 0.1 and a punishment rate of 0.05. These results are shown in Table 5. Since the punish-

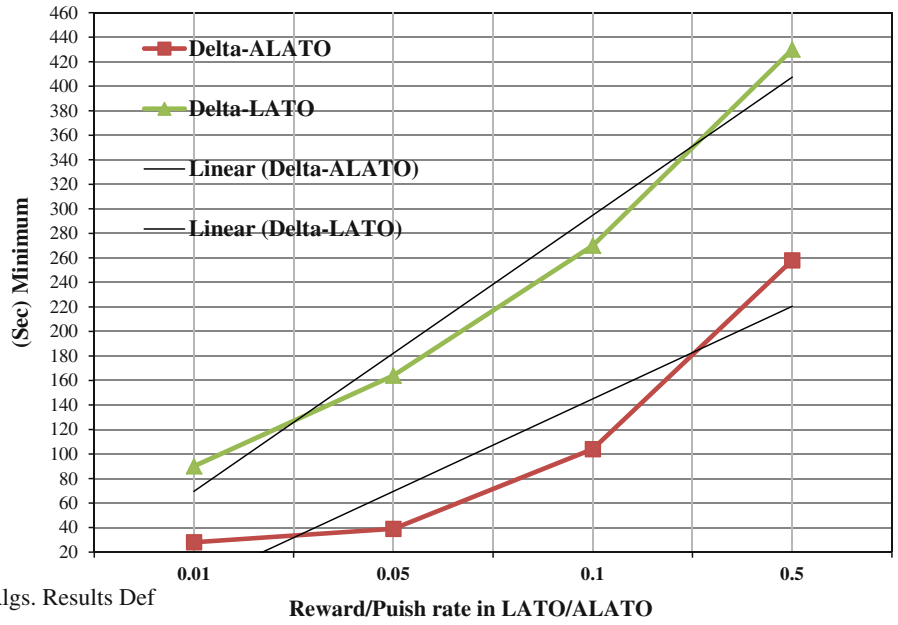

Fig. 7 Learning trend for the ALATO and LATO algorithms

Algs. Results Def

Table 5 Changes in punishment and reward rates in the S-L_{RP} model and in time optimization algorithms with a fixed budget of 150,000

(a,b)	State 1	State 2	State 3	State 4
Algorithms	(0.01, 0.01)	(0.05, 0.05)	(0.1, 0.1)	(0.5, 0.5)
BTO	17,440	17,900	17,950	18,543
AEBTO	17,240	17,440	17,645	17,840
LATO	17,230	17,220	17,440	17,460
ALATO	17,168	17,160	17,370	17,400
Minimum	17,150	17,150	17,150	17,150

ment and reward rates are equal they should be small so that the algorithm can gradually reach the appropriate schedule. For a fixed budget of 150,000, the punishment and reward rates are the same, and 0.05 is the best punishment/reward rate for LA-based algorithms.

In this model, the desired environment response depends on the low cost of the selected resource relative to the previous iteration, and so the results of the S-L_{RP} model are better than the results of the P-L_{RP} model.

Figure 8 shows the execution times in acceptable states of the S-L_{RP} automata and in a state where the punishment and reward are 0.05. As can be seen, the ALATO algorithm outperforms the LATO algorithm by about 60 time units, and its difference from the optimal state is only 10 time units.

S-L_{RI} model

In this model, the punishment rate is 0 ($b = 0$), and four different values were tested for the reward rate. The results are shown in Table 6. Since the punishment rate is 0, tasks that select the incorrect resource are not punished and the LAs cannot direct the proposed algorithm toward the appropriate schedule.

S-L_{ReP} model

In this model, the punishment rate is much smaller than the reward rate, and four different values for punishment and reward were tested. The results are shown in Table 7.

Table 7 shows that the best punishment and reward rates for S-L_{ReP} are $a = 0.1$ and $b = 0.05$. Considering that the desired environment response to the selected operation (αi) in the nth iteration is proportional to the low cost of the selected resource relative to the previous iteration and that the reward is 0.1, the less expensive the resource the more probable it is that the operation and algorithm are directed toward better scheduling that optimizes cost.

Figure 9 compares the automata in the LATO and ALATO algorithms and compares both with the Minimum algorithm. Here the best execution time for each automaton for a budget of 150,000 is examined. The ALATO algorithm is much more improved than the LATO algorithm from the point of view of execution time and is therefore close to the optimal state. The S-L_{ReP} model uses less time than the other automata because the greater the reward-to-punishment ratio, the greater the possibility of updating and selecting appropriate resources with fewer iterations. The S model has thus yielded better results than the P model. This is because by considering more detailed changes, the precision of the automata is increased and convergence to fixed probabilities for resource selection is reached in less iteration. As a result, the necessary time for stabilizing the automata is greater than for the P model.

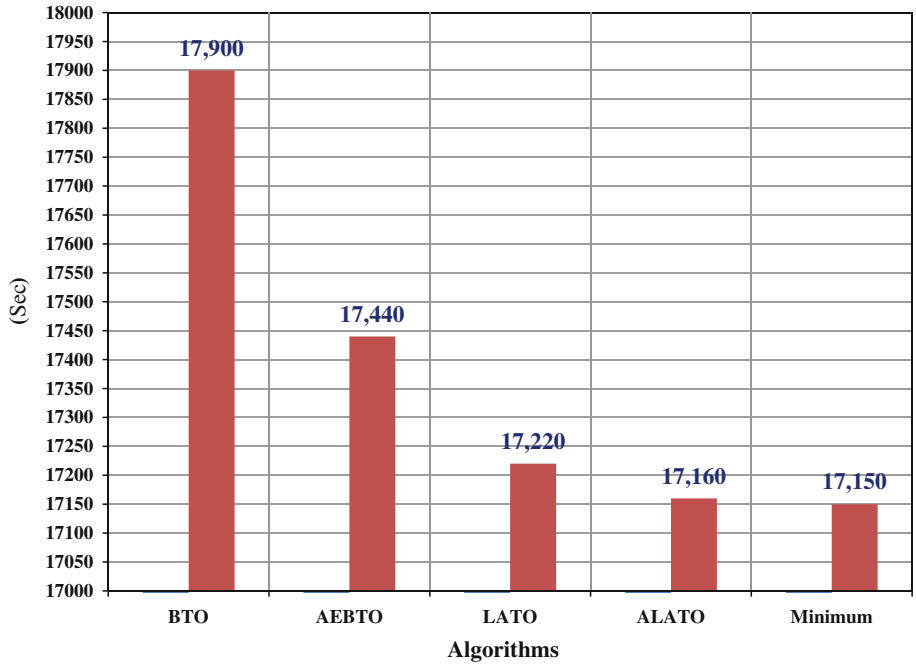

Fig. 8 Comparison of execution times of algorithms in S-L_{RP} automata with fixed budget of 150,000

Table 6 Changes of punishment and reward rates in S-L_{RI} model and in time optimization algorithms with fixed budget of 150,000

(a,b)	State 1	State 2	State 3	State 4
Algorithms	(0.01, 0.01)	(0.05, 0.05)	(0.1, 0.1)	(0.5, 0.5)
BTO	17,440	17,900	17,950	18,543
AEBTO	17,240	17,440	17,645	17,840
LATO	17,238	17,314	17,420	17,580
ALATO	17,178	17,275	17,354	17,408
Minimum	17,150	17,150	17,150	17,150

Table 7 Changes of punishment and reward rates in S-L_{ReP} model and in time optimization algorithms with fixed budget of 150,000

State 4	State 3	State 2	State 1	(a,b)
(0.5, 0.5)	(0.1, 0.1)	(0.05, 0.05)	(0.01, 0.01)	Algorithms
18,543	17,950	17,900	17,440	BTO
17,840	17,645	17,440	17,240	AEBTO
17,470	17,383	17,210	17,232	LATO
17,440	17,365	17,158	17,183	ALATO
17,150	17,150	17,150	17,150	Minimum

Time optimization within specified time with different budgets (case II)

In this case, scheduling algorithms with the aim of time optimization are compared for different budgets. In addition, it is shown how much time each of the algorithms takes to complete execution of the user applications under the same conditions (users, applications, and resources). Obviously, the algorithm that schedules tasks for the resources so that the application results are delivered to the user in the shortest time is more appropriate for the grid environment.

We tested ALATO with various models of learning automata under the conditions listed in Table 8. The task lengths in all models in the tests were considered equal, in order to have appropriate comparisons, and there was a uniform distribution of task lengths within the task length range. The "number of tests" parameter shows that 20 experiments were performed to obtain an execution time within a specified deadline for an application using the available algorithms, and their mean values were selected for examination.

S-model automata were selected for the study because they yield better execution times than standard automata. Among the various *S* models, S-L_{ReP} automata yielded the best results. Therefore, the results are assessed only for these automata. Table 9 shows the results of experiments comparing time optimization algorithms for the S-L_{ReP} model automata that yielded the best time. The first column shows the budget specified by the user. The budget is changed from 130,000 to 180,000 in order to assess and compare the efficiency of the algorithms with different budgets. The other columns represent each of the algorithms, and the last column shows the Minimum algorithm's execution time for the different budgets.

In most cases, the minimum time was not realized by the algorithms. Each table cell shows the execution time of the application for the specified scheduling algorithm and budget. For example, for a budget of 150,000, the BTO scheduling algorithm completed execution of the user application after 17,900 time units, while the LATO and ALATO algo-

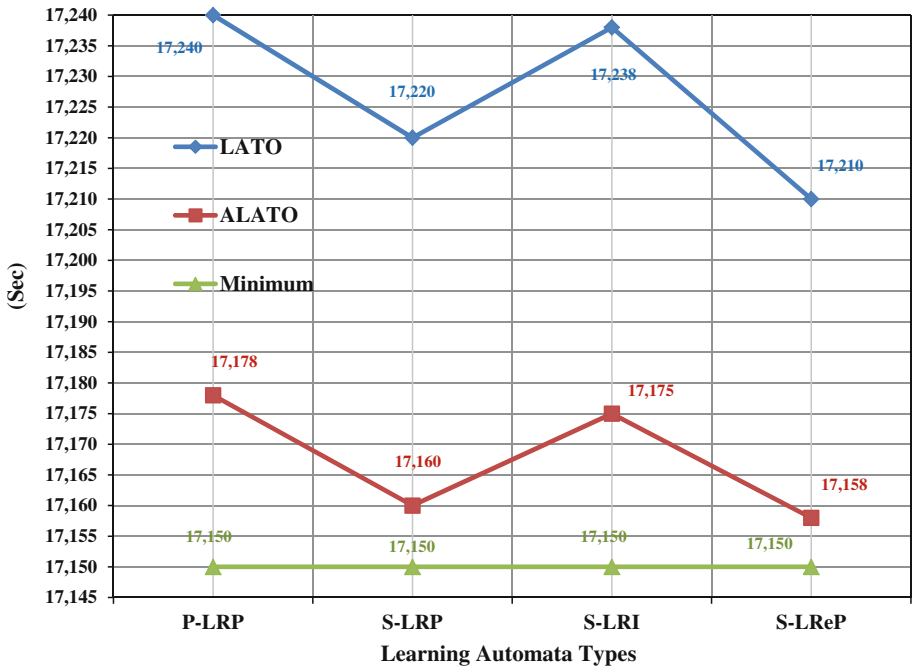

Fig. 9 Comparison of execution times of automata with fixed budgets of 150,000 obtained in the best state of each automata

Table 8 Time optimization conditions with variable budgets

Parameter	Value	Parameter	Value
Algorithm	Alternate	Deadline	50,000
User number	1	Budget	Alternate
Task no.	200	Task length range	[10...200]
Resource configure	GR1	Test no.	20

Table 9 Execution time for applications in time optimization algorithms with different budgets

Algorithms					Budget
Minimum	ALATO	LATO	AEBTO	BTO	
28,020	28,200	28,220	28,220	28,300	130,000
21,250	21,305	21,370	21,450	21,500	140,000
17,150	17,158	17,210	17,440	17,900	150,000
14,300	14,305	14,400	14,490	14,610	160,000
12,620	12,642	12,750	12,760	12,890	170,000
11,090	11,120	11,220	11,270	11,320	180,000

rithms required 17,210 and 17,158 time units, respectively. Twenty tests were performed for each budget and the average of the resultant values for each algorithm was considered as the mean time for task execution with the specified budget. Note that while it is possible to compare algorithms with a budget of 140,000 units, for example, we cannot place the resultant values next to values resulting from another budget such as 150,000 and meaningfully study the changes.

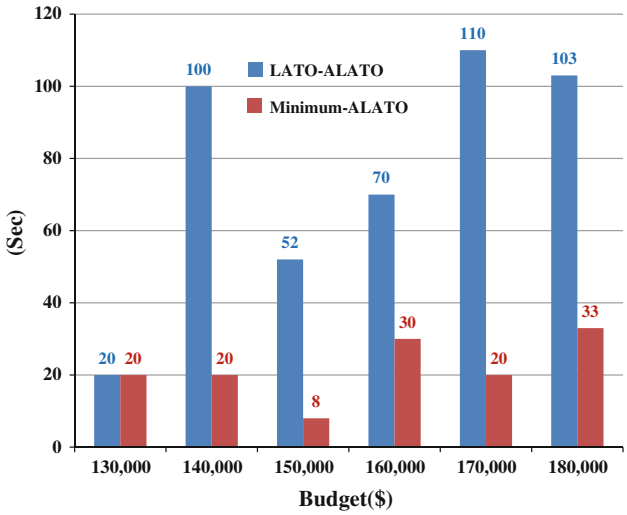

Fig. 10 Comparison of ALATO, LATO, and minimum algorithms with different budgets

Different values are obtained for the task execution times for each of the algorithms; hence, we had to calculate the average task execution time for each algorithm after several experiments. The differences are due to the random distribution of task lengths within the specified range, and the more heterogeneous the tasks (the wider the range of the task length distribution), the greater are these differences.

Here the two intelligent algorithms (LATO and ALATO) are compared. Figure 10 shows the results for these two algorithms using different budgets. It can be seen that the execution times resulting from the ALATO algorithm are less than for the LATO algorithm. The ALATO algorithm can save up

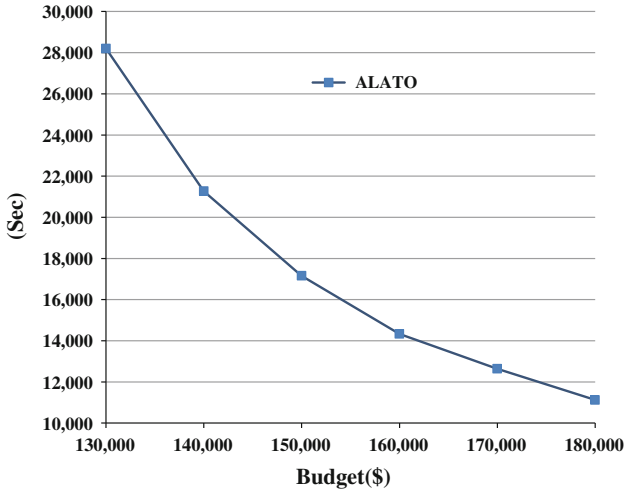

Fig. 11 Trend of changes in task execution times for the ALATO algorithm with different budgets

to 100 time units for small budgets, while the time saving for large budgets is approximately 110 units.

To see that the improvement made possible by the ALATO algorithm renders it the best proposed algorithm, ALATO should be compared with the Minimum algorithm's execution times for the tasks. Figure 10 shows the difference between the results obtained by the ALATO algorithm and the minimum execution times for different budgets. It can be seen that this difference is insignificant and is less than 8 units in all cases. When the execution time of tasks for all budgets is greater than 10,000 units, this amount is negligible. Therefore, it can be said that the ALATO algorithm produces as close to the minimum execution time as is possible, and since we know that in most cases this minimum is not attainable by any algorithm, a new algorithm that can yield at most 8 units of improvement is not worth considering.

As was seen earlier, all proposed algorithms have the same trend in the ratio of execution time to budget. For example, the trend for the LATO algorithm is shown in Fig. 11. The most important aspect of these changes is the decrease of task execution time as a result of budget increases. This happens because when the proposed budget is increased; user applications can use more resources to execute their tasks, including more expensive resources, and thus decrease their execution time. This trend of decreasing time can be seen for all algorithms. Indeed, the decrease is most rapid at the outset, while the greater the budget is, the less is the decrease, so that the change in execution time with respect to the budget is nonlinear. Initially, when the budget is small, the algorithms should use a small number of inexpensive resources, so that the execution cost is less than the budget; but as the budget increases, the algorithms can use more expensive resources for task execution and thus reduce execution time. Obviously, additional resources gradually become more expensive, and as these expensive resources are used more, budget increases have smaller impacts on reducing execution time. Eventually, when all resources have been used for task scheduling for some specified budget, budget increases make no difference to execution time and the graph becomes a horizontal line. In contrast, the initial point of the graph has the smallest budget for delivering the tasks within the deadline, which is of course the minimum cost for the computations. For the deadline of 100,000 units used in these tests, the minimum cost was computed as 115,000 units, and so we started our experiments with a larger budget of 120,000 units.

Time optimization for different heterogeneities (case III)

In this case, we examined our proposed algorithm along with *S*- model LAs for all four resources. We tested *S*-model LAs with different values for the punishment and reward rates. Among the *S*-model methods, as will be seen, the best results for an operation selected in the *n*th iteration were obtained by the S-L_{ReP} method with a reward rate of 0.1 and a punishment rate of 0.05. Every task has a unique length specified in terms of MI units. Task lengths are always considered variable within the selected (*minimum … maximum*) range. A value is randomly selected from this range for each task length, so that the distribution of task lengths is uniform. For the most homogeneous state, the range of task lengths was chosen to be (100,000 … 110,000), and for the most heterogeneous state, it was chosen to be (10,000 … 200,000). The simulation conditions are listed in Table 10. The task length range is variable, and the experiments are performed for different ranges (i.e., different heterogeneities).

Twenty experiments were performed to obtain the execution time values for the algorithms in the heterogeneous states, and the average values were selected for examination.

Since the S-model automata yield better time than standard automata, these automata were selected for study. Of the various *S* models, the S-L_{ReP} automata yielded the best results, so only the results for these automata are examined. Twenty tests were performed for each heterogeneity, and the average value obtained for each of the algorithms was considered to be the average execution time for the tasks.

Table 10 Conditions for time optimization experiment for different heterogeneities and budget of 150,000 (in sec)

Parameter	Value	Parameter	Value
Algorithm	Alternate	Deadline	20,000
User number	1	Budget	150,000
Task no.	200	Task length range	Alternate
Resource configure	GR1	Test no.	20

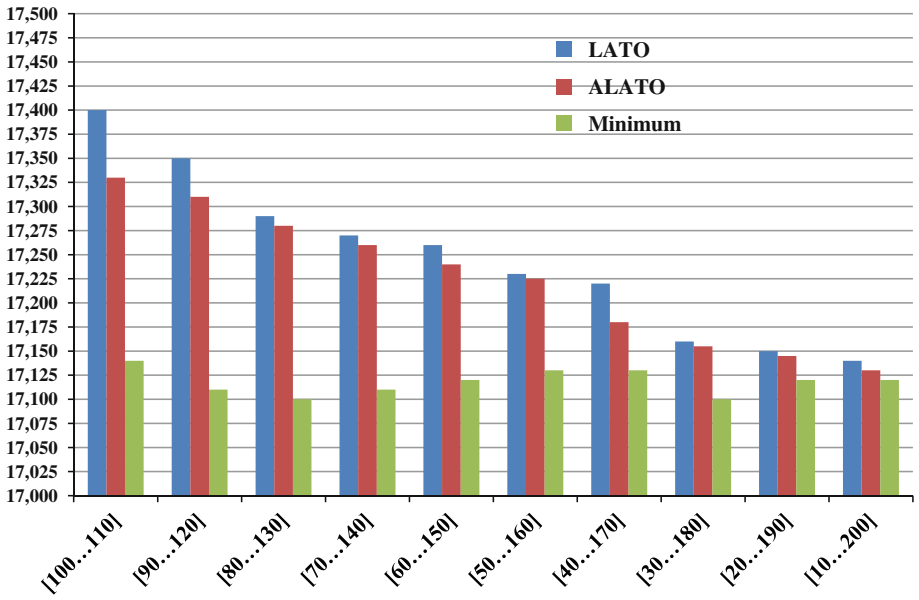

Fig. 12 Comparison of minimum, LATO and ALATO algorithms for different heterogeneities

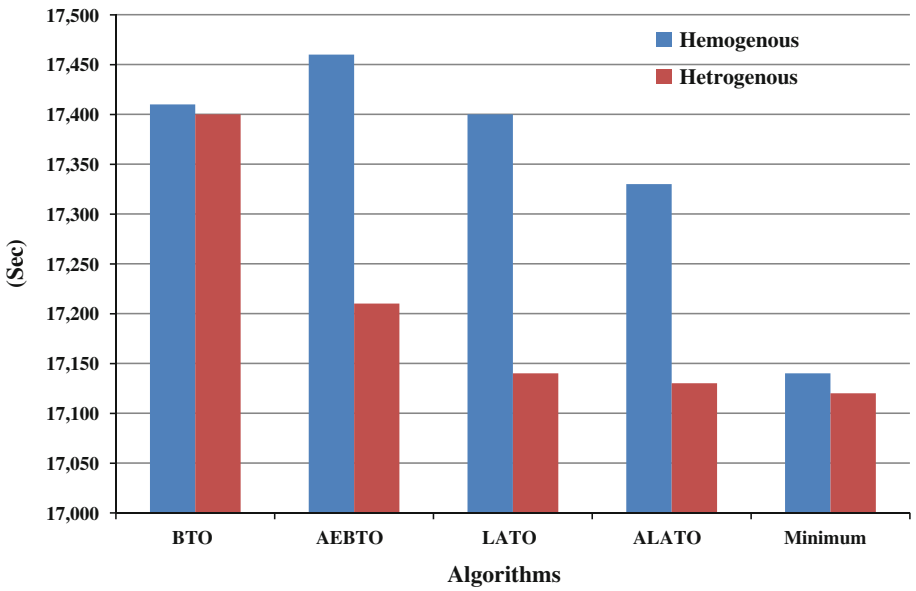

Fig. 13 Results of time optimization algorithms for homogeneous and heterogeneous tasks

The LATO and ALATO algorithms use LAs, and the ALATO algorithm improves the results of the LATO algorithm. Here we study the effect of the heterogeneity of tasks on the improvements achieved by the LATO and ALATO algorithms.

Figure 12 shows the results for these two algorithms, along with the results of the Minimum algorithm, for different heterogeneities. It can be seen that the improvement achieved by LATO and ALATO scheduling increases with the heterogeneity of the tasks. The time reduction trend for ALATO is faster than for LATO, and the speed with which ALATO reaches the minimum execution time is greater than it is for LATO.

As can be seen, ALATO comes closer to the optimum (Minimum) algorithm for high heterogeneities than does LATO.

Figure 13 shows the final results achieved by the time optimization algorithms for two states, one homogeneous and the other heterogeneous. In the homogeneous state, the range of the uniform and random distribution of task lengths is (100 … 110), and in the heterogeneous state it is (10 … 200). Except for the BTO algorithm, which has almost the same performance in the two states, the other algorithms achieved some improvement in the heterogeneous state compared to the homogeneous state. This improvement is more significant for LATO and ALATO than for the other algorithms. Also,

it can be seen that results obtained from the algorithms in the homogeneous state are closer to each other than in the heterogeneous state, where the execution times for tasks are very different for the different algorithms.

Conclusion

This study proposed a new ASPN that uses LAs to improve execution time in an economic grid. The proposed ASPN falls into the fusion hybrid category. The new design is quite flexible in dynamic environments such as grid networks, because LAs do not require training with previous data. In addition, every token in an LA-based ASPN has its own automaton, and there is no need to add new transitions and places, both of which make the design of the net computationally simple, with no increase in computation overload. Through our simulated time optimization algorithms, we showed that the ALATO learning algorithm outperforms exploratory algorithms for highly heterogeneous tasks and executes user applications on the grid within a specified budget in shorter time than the other algorithms. In the sequel, we conclude that the performance of ALATO is higher than the existing algorithms in the proposed examples. In general, Therefore, we are able to apply this method in the grid systems that are similar to the cases we have tested in the test bed.

Our planned future work will address the new concept of insurance. We will consider insurance for time delays, with a user receiving compensation when there is a delay in the delivery of the application results. These delays can have different causes, such as network failures, resource failures, and other possible events about whose occurrence the system has no prior information. A grid can insure a network and different resources and equipment against these events through insurance agencies and, in turn, charge clients insurance fees to protect them from possible delays. To this end, a comprehensive analysis of the events that may affect the execution of tasks in the grid should be made. Through an examination of previous data, insurance costs, and possible compensations, it is possible to compute an increase in execution costs, and this amount can be charged to clients who request insurance against some or all delaying events.

References

Aissani, N., Bekrar, A., Trentesaux, D., & Beldjilali, B. (2011). Dynamic scheduling for multi-site companies: A decisional approach based on reinforcement multi-agent learning. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0580-y.

Al-Ali, R., Rana, O., Walker, D., Jha, S., & Sohail, S. (2002). G-QOSM: Grid service discovery using QOS properties. *Computing and Informatics Journal, Special Issue on Grid Computing*, *21*(4), 363–382.

Al-Khasawneh, A., & Bsoul, M. (2010). Job scheduling in economic grid environments. *International Journal of Information and Communication Technology, 2*(3), doi:10.1504/IJICT.2010.032410.

Arab, A., Ismail, N., & Lee, L. S. (2011). Maintenance scheduling incorporating dynamics of production system and real-time information from workstations. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0616-3.

Archimede, B., Letouzey, A., Memon, M. A., & Xu, J. (2013). Towards a distributed multi-agent framework for shared resources scheduling. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-013-0748-8.

Assuncao, M., & Buyya, R. (2006). An evaluation of communication demand of auction protocols in grid environments Technical report, University of Melbourne, Melbourne, Australia.

Baranauskas, V., Bartkevičius, S., & Šarkauskas, K. (2006). Coloured Petri nets—tool for control systems learning. *Electronics and Electrical Engineering.*, *4*, 41–46.

Bilyk, A., & Mönch, L. (2010). A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-010-0464-6.

Boppana, R., & Halldorsson, M. M. (1992). Approximating maximum independent sets by excluding subgraphs. *BIT Magazine, Published By BIT Computer Science and Numerical Mathematics*, *32*(2), 180–196. doi:10.1007/BF01994876.

Bui, T. N., & Eppley, P. H. (1995). A hybrid genetic algorithm for the maximum clique problem. In *Proceedings of 6th international conference on genetic algorithms*, San Francisco, CA, USA, pp. 478–484.

Buyya, R. (2002). Economic-based distributed resource management and scheduling for grid computing, Ph.D. Thesis, School of Computer Science and Software Engineering, Monash University, Melbourne, Australia.

Buyya, R. (2002). The World-Wide Grid (WWG). http://www.buyya.com/ecogrid/wwg/.

Czajkowski, K., Foster, I., Kesselman, C., Sander, V., & Tuecke, S. (2002). SNAP: A protocol for negotiating servi^{2}ce level agreements and coordinating resource management in distributed systems. In *8th workshop on job scheduling strategies for parallel processing*, pp. 153–183.

Fleszar, K., Charalambous, C., & Hindi, K. S. (2011). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0522-8.

Guinet, A. (1995). Scheduling independent jobs on uniform parallel machines to minimize tardiness criteria. *Journal of Intelligent Manufacturing*, *6*, 95–103.

Hirel, C., Wells, S., Fricksy, R., & Trivedi, K. S. (2000). ISPN: An integrated environment for modeling using stochastic petri nets. In *Center for advanced computing and communication department of electrical and computer engineering Duke University*. Durham, NC 27708–0291.

Jain, L. C., & Martin, N. M. (1998). *Fusion of neural networks, fuzzy sets, and genetic algorithms: Industrial applications, 1st ed.*. FL, USA: CRC Press ISBN: 0849398045.

Jeng, M. D., Lin, C. S., & Huang, Y. S. (1999). Petri net dynamics-based scheduling of flexible manufacturing systems with assembly. *Journal of Intelligent Manufacturing, 10*, 541–555. doi:10.1023/A:1008960721370.

Khosla, R., & Dillon, T. (2002). Intelligent hybrid multi-agent architecture for engineering complex systems. *International Conference on Neural Networks, 4*, 2449–2454. doi:10.1109/ICNN.1997.614540 Houston, TX.

Mahdavifar, Y., & Meybodi, M. R. (2007). Cost-time optimization in economic computational grids. In *Proceedings of the third informa-*

tion and knowledge technology, Mashad, Iran: Ferdowsi University of Mashad.

Mahdavifar, Y., & Meybodi, M. R. (2007). Time optimization in economic computational grids using learning automata. In *Proceedings of the first Iranian data mining conference*. Tehran, Iran: Amirkabir University of Technology.

Mirzaee, A., & Rahimzadeh, P. (2011). A agent-based decentralized algorithm for resource semantic discovery in economic grid. In *IEEE 3rd international conference on communication software and networks (ICCSN)*, pp. 306–311. doi:10.1109/ICCSN.2011.6013721.

Moore, J., & Hahn, L. (2003). Petri net modeling of high-order genetic systems using grammatical evolution. *Bio Systems*, *72*(2), 177–186. doi:10.1016/S0303-2647(03)00142-4.

Murata, T. (2002). Some recent applications of high-level Petri nets. *IEEE International Symposium on Circuit and System*, *2*, 818–821. doi:10.1109/ISCAS.1991.176488.

Narendra, K. S., & Thathachar, M. A. L. (1989). *Learning automata: An introduction*. Englewood Cliffs, NJ, USA: Prentice-Hall, ISBN 0134855582.

Pasandideh, S. H. R., Niaki, S. T. A., & Hajipour, V. (2011). A multi-objective facility location model with batch arrivals: Two parameter-tuned meta-heuristic algorithms. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0592-7.

Peterson, J. (1981). *Petri net theory and the modeling of systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, ISBN 0136619835.

Pla, A., Gay, P., Meléndez, J., & López, B. (2012). Petri net-based process monitoring: A workflow management system for process modelling and monitoring. *Journal of Intelligent Manufacturing*, doi:10.1023/A:1012292102123.

Pooranian, Z., Shojafar, M., Abawajy, J. H., & Abraham, A. (2013). An efficient meta-heuristic algorithm for grid computing. *Journal of Combinatorial Optimization (JOCO)*, doi:10.1007/s10878-013-9644-6, Springer.

Pooranian, Z., Shojafar, M., & Javadi, B. (2012). Independent task scheduling in grid computing based on queen bee algorithm. *IAES International Journal of Artificial Intelligence (IJ-AI)*, *1*(4), 171–181. doi:10.11591/ij-ai.v1i4.1229.

Pooranian, Z., Shojafar, M., Abawajy, J. H., & Singhal, M. (2013b). GLOA: A new job scheduling algorithm for grid computing. *International Journal of Interactive Multimedia and Artificial Intelligence (IJIMAI)*, *2*(1), 59–64. doi:10.9781/ijimai.2013.218.

Pooranian, Z., Shojafar, M., Tavoli, R., Singhal, M., & Abraham, A. (2013b). A joint meta-heuristic algorithm applied in job scheduling on computational grids. *Informatica*, *37*(2), 157–164.

Poznyak, A. S., & Najim, K. (1997). *Learning automata and stochastic optimization*. NY: USA: Springer, ISBN: 3540761543.

Radakovič, M., Obitko, M., & Mařík, V. (2011). Dynamic explicitly specified behaviors in distributed agent-based industrial solutions. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-011-0593-6.

Reddy, S. R. (2006). Market economy based resource allocation in grids Master's thesis. Indian Institute of Technology, Kharagpur, India.

Reisig, W. (1985). *Petri nets: An introduction, EATCS monographs on theoretical computer science*. USA: Springer. ISBN: 3642699707.

Sarhadi, A., & Meybodi, M. R. (2010). New algorithm for resource selection in economic grid with the aim of cost optimization using learning automata. *International Conference on Challenges in Environmental Science and Computer Engineering (CESCE)*, *1*, 32–35. doi:10.1109/CESCE.2010.185.

Schwardy, E. (2001). *Optimization of Petri nets structure using genetic programming*. Dept. of Cybernetics and Artificial Intelligence. Faculty of Electrical Engineering and Informatics. University of Technology Koice. Slovakia.

Shojafar, M., Barzegar, S., & Maybodi, M. R. (2011). Time optimizing in economical grid using adaptive stochastic Petri net based on learning automata. In *Proceedings of International Conference on Grid Computing & Applications (GCA), WORLDCOMP*, pp. 67–73.

Shojafar, M., Pooranian, Z., Abawajy, J. H., & Meybodi, M. R. (2013). An efficient scheduling method for grid systems based on a hierarchical stochastic Petri net. *Journal of Computing Science and Engineering (JCSE)*, *7*(1), 44–52. doi:10.5626/JCSE.2013.7.1.44.

Venkataramana, R. D., & Ranganathan, N. (1999). Multiple cost optimization for task assignment in heterogeneous computing systems using learning automata. *Heterogeneous Computing Workshop (HCW'99), IEEE Computer Society*, pp. 137–145. doi:10.1109/HCW.1999.765118.

Zhou, M. C., & Jeng, M. D. (2002). Modeling analysis simulation scheduling and control of semiconductor manufacturing systems: A Petri net approach. *IEEE Transaction on Semiconductor Manufacturing, 11*(3), 333–357. doi:10.1109/66.705370, ISSN: 0894–6507.

Zimmermann, A., Rodriguez, D., & Silva, M. (2001). A two phase optimization method for Petri net models of manufacturing systems. *Journal of Intelligent Manufacturing*, *12*, 409–420.