A Coevolutionary Approach to Cellular Automata-Based Task Scheduling

Gina M.B. Oliveira and Paulo M. Vidica

Faculdade de Ciencia da Computacao - Universidade Federal de Uberlandia (UFU)
Av. Joao Naves de Avila, 2121- Campus Santa Monica, Bloco B, sala 1B60
CEP: 38400-902 Uberlandia, MG, Brazil
gina@facom.ufu.br

Abstract. Cellular Automata (CA) have been proposed for task scheduling in multiprocessor architectures. CA-based models aim to be fast and decentralized schedulers. Previous models employ an off-line learning stage in which an evolutionary method is used to discover cellular automata rules able to solve an instance of a task scheduling. A central point of CA-based scheduling is the reuse of transition rules learned for one specific program graph in the schedule of new instances. However, our investigation about previous models showed that evolved rules do not actually have such generalization ability. A new approach is presented here named multigraph coevolutionary learning, in which a population of program graphs is evolved simultaneously with rules population leading to more generalized transition rules. Results obtained have shown the evolution of rules with better generalization abilitywhen they are compared with those obtained using previous approaches.

Keywords: Cellular automata, task scheduling, coevolution.

1 Introduction

Scheduling tasks in multiprocessor architectures is known to be a NP-complete problem [4] and it is still a challenge in parallel computing field. Approaches to explore task scheduling typically employ specific heuristics [11] or metaheuristics, like genetic algorithms and simulated annealing [12]. A computational effort is used to solve an instance of the problem in these approaches and when a new instance is presented to the algorithm, the process needs to start again from scratch. Besides, the majority of scheduling algorithms is sequential and they are not appropriate to be implemented in parallel hardware. Promising results obtained with the join use of genetic algorithms (GA) [1] and cellular automata [3] have shown a new perspective direction in developing fast and parallel scheduling algorithms [4-8].

Cellular automata (CA) are discrete dynamical systems that consist of a large number of simple components with local connectivity. It is necessary to design an appropriate neighborhood structure to use CA for scheduling. This neighborhood must reflect the structure of a program graph, which contains all the relevant information of the parallel application such as precedence of tasks, their computational costs and communication costs between pairs of tasks. In the present work a nonlinear structure named *Selected Neighborhood* [4] is used to capture the intrinsic characteristics of program graphs. CA model applies an asynchronous updating of cells since it has returned better results in previous works [4, 10].

Some CA-based approaches had presented good results when applied to schedule tasks for given program graphs [4, 8]. These approaches have a learning phase characterized by the use of a standard GA to find the correct distribution of the tasks considering a single program graph. The population of such GA is formed by CA transition rules aiming to schedule tasks over architecture processors. Later on, evolved CA rules can be used to schedule the same program graph or they can be applied to find optimal or suboptimal solutions to other parallel programs. However, as we will show in Section 4, the rules evolved using previous schemes [4, 8] have shown almost none generalization ability, that is, they cannot find reasonable makespam for new program graphs not seen during learning, even if these instances are very similar to the one used to find them.

A new approach is proposed here, in which a coevolutionary algorithm is used during learning to search for good scheduling not only for the target program graph but also for some similar graphs generated applying some mutations on the target graph. The goal is to find CA rules with better generalization ability. Therefore, a second population formed by variations of the target graph is evolved simultaneously with the transition rule population. We call this new approach as multigraph coevolutionary learning.

Seredynski and Zomaya [5] have also investigated a coevolutionary approach in the learning phase of CA-based schedulers. However, in their approach the second population was formed by different initial configurations of the CA lattice to be used as initial allocations to evaluate rules for a single program graph. The goal of coevolutionary search in [5] is not to improve the generalization ability of the discovered rules as we are seeking here, but to improve the performance of these rules in relation to the proper target program. Other related works investigated CA models to task scheduling [7, 13, 14]. However, they employed a different kind of neighborhood (linear), while selected neighborhood used here and in references [4, 5, 6, 8] have shown more efficient to solve scheduling.

The paper is organized as follows. Section 2 contextualizes the problem of scheduling tasks on multiprocessor systems. Section 3 reviews previous CA models on scheduling. Section 4 discusses the generalization ability of CA rules – a key point of this work. Section 5 presents the multigraph coevolutionary learning proposed here. Section 6 presents some experiments accomplished with the new approach. Section 7 shows the main conclusions of this work.

2 Multiprocessor Task Scheduling

A parallel application is represented by a program graph which is a weighted directed acyclic graph $G_p = (V_p, E_p)$. V_p is the set of N_p tasks of the parallel program. It is assumed that each task is a computational indivisible unit. There is a precedence constraint relation between tasks k and l if the result produced by task k has to be available to task l, before it starts. E_p is the set of precedence relations between tasks. A program graph has weights associate to nodes and edges: b_k describes the processing time required to execute a given task on any processor and a_{kl} describes the communication time between pairs of tasks k and l, when they are located in distinct processors. The purpose of scheduling is to distribute the tasks of a parallel program among the processors in such a way that the precedence constraints are preserved and the total execution time (or makespam) T is minimized. T depends on tasks allocation between processors and on some scheduling policy that defines the order for executing tasks within processors. Attributes can be associated with each node k in a program graph such as: the level h_k is defined as the maximal length of the longest path from a node k to an exit node and the co-level d_k is defined as the length of the longest path from the starting node to node k. If they are calculated depending on tasks allocation they are called dynamic level and dynamic co-level [4]. Some special sets are defined in relation to a given task k: *predecessors(k)*, *brothers(k)* (nodes that have at least one common predecessor) and *successors(k)*. The multiprocessor architecture used in the present work is composed by two identical processors: it is assumed that both processors have the same computational power and communications between channels do not consume any extra time.

3 Cellular Automata-Based Scheduling

Cellular automata are discrete complex systems that possess both a dynamic and a computational nature. Basically, a cellular automaton consists of two parts: the cellular space and the transition rule. Cellular space is a regular lattice of N cells, each one with an identical pattern of local connections to other cells, and subjected to some boundary conditions. CA are characterized by a transition rule, that determines which will be the next configuration of the lattice, from their current one. Cells interact locally in a discrete time t: for each cell i, a neighborhood of radius r is defined; the state of the cell i at time $t + 1$ depends only on the states of its neighborhood at time t and its current state. Cells updating is usually performed in a synchronous way. However, asynchronous updating is also possible. In sequential updating, only one cell is update at each time step, starting from the first cell.

A scheduler model was presented in [4] where each lattice cell is associated with a task. Considering two-processor architectures, the lattice is binary and each cell state indicates that the corresponding task is allocated either in the processor P0 or P1. Each lattice configuration corresponds to a different allocation of the tasks in the processors. Makespam T associated to a given lattice is calculated using a scheduling policy that defines the order of the tasks within each processor. The scheduling policy used is: the task with the highest dynamic level is performed first.

CA evolve according to their transition rules. The goal is to find a CA rule able to converge the lattice to a final allocation of tasks, which minimizes T, starting from any initial allocation. The nonlinear structure of a program graph is represented in [4] by a nonlinear CA neighborhood named Selected Neighborhood. Only two selected representatives of each set of predecessors, brothers and successors are used to create the neighborhood of a cell associated with a task *k* [4]. The two representative tasks are selected on the basis of maximal and minimal values of a chosen attribute. The neighborhood for a cell *k* consists of 7 cells (including cell *k)*. The length of a rule is 250 bits and there are 2^{250} possible rules [4]. Although sequential updating of cells is not usual as the synchronous one, considering CA broad context, it has returned interesting results in previous works involving CA-based scheduling [4, 6] and it is also investigated in the present work.

CA-based scheduler presented in [4] has two stages: a learning phase and an operating phase. In the learning phase, a GA [1] is used to search for CA rules able to schedule a specific program graph. GA begins with a population of *P* rules randomly generated, encoded as binary strings. Rule's fitness is calculated by: (i) using a set of initial configurations (ICs) corresponding to initial allocations of a program graph; (ii) applying the rule starting from each IC for *M* time steps and calculating the makespam *T* for each final allocation; (iii) adding the values of *T* calculated for all the ICs evaluated to obtain the fitness. The smaller is the fitness the better is the rule. GA is simple and it is based on the framework to evolve rules to solve density classification task [12]. At the end, population is stored in a repository of rules. In operating phase, it is expect that, for any initial allocation of tasks, stored rules will be able to find an allocation providing the optimal *T* (or near to). It is also expected that rules evolved in the learning phase can be used in other graphs different from the one for which they were learned providing a good performance. The scheduler model in [4] was further investigated in references [5, 6, 8].

4 Generalization Ability of Previous Models

In previous CA-based schedulers [4, 8], a single program graph was used to discover CA rules in learning phase. GA evolves the rules calculating their fitness based only on this target graph. Published results have shown that the CA-based scheduler was able to discover rules that can successfully schedule several program graphs investigated in literature. However, when the discovered rules evolved for a specific graph were applied to other programs, conclusive results about the generalization ability of rules have not been published as we argument in the following.

It is interesting to point out that this generalization ability of evolved rules is crucial to CA-based scheduling. The huge computational effort necessary to discover the CA rules is justified only if these rules are able to be reused in new problems. Otherwise, a GA can be directly used in the search for optimal configurations of each graph independently, without the need to involve CA rules in the model. The idea behind the application of transition rules is the possibility of reusing them in new instances, without the need of a new process of evolutionary learning.

The generalization ability was evaluated in [4] using graphs Gauss 18 and Tree15. It was presented that rules evolved for Gauss18 were successfully applied to schedule Tree 15. However, when the rules evolved for Tree15 were applied to schedule Gauss 18, the results were not good: the best makespam found was 62 and the optimum is 44 (time units). Besides, the rules evolved for Tree 15 in [4] were applied to "similar" graphs Tree63 and Tree127. In this last test, the rules were able to schedule the two new graphs obtaining the optimum time. What is important to point here is that the family Tree*x* is formed by graphs with a very regular structure (binary trees), which are easy to schedule. We performed a simple test to evaluate the difficulty of solving the scheduling of Tree15: 10,000 random lattices of CA were sorted and *T* associated to each allocation was calculated. The optimum *T* is 9: average *T* found was close to 9.8 and in approximately 40% of the random allocations optimum was obtained. So, even using a random allocation it is relatively easy to obtain the best scheduling for this kind of graph. The same test was performed for Gauss18, which has the optimal makespam equal to 44. In this case the average makespam obtained was close to 75 and none of the 10,000 random allocations returns the optimum. It is clear that for Gauss18, which represents a real parallel program, the scheduling problem is actually difficult to solve. Therefore, the results obtained reusing rules evolved for Gauss18 in the scheduling of Tree15 presented in [4] must be taken in a relative sense: the results are as good as if one uses a random allocation.

In subsequent works, few results about the generalization ability were presented. In [5] the generalization results refer again to the application of rules in family Tree*x*. In [6] this generalization ability was tested using rules evaluated for two program graphs separately (g18 and rnd25) in a new graph formed by joining these two graphs. Using a model based on an artificial immune system, the rules were able to schedule the mixed graph. Although these results are interesting, the learning phase was applied again to obtain the new rules. The related studies [4-8] are promising but just a little information about generalization ability was shown.

Aiming to better investigate such desirable ability, we realized experiments based on the model described in [4] and [8] applying the evolved rules to graphs totally different from the one used in the learning phase. In these experiments we used four graphs also applied in [4-8]: Gauss18, G18, Tree15 and G40 with optimum equal to 44, 46, 9 and 80, respectively. Each one of these graphs was used in learning phase as the target graph and then evolved rules were applied to the target graph and to the other three graphs in the operating phase. It was possible to observe that for each evaluated target graph the best rule evolved is able to find the optimal makespam for all the initial configurations tested. However, when the best evolved rules were applied to a graph different from the target one the results were not good in general. The unique exception is graph Tree15: the average makespam (considering all IC) when the target graph used as an input for the learning phase was Gauss18, G18 and G40 is 9.84, 9.63 and 9.39, respectively. For the other graphs, the results are not reasonable. For Gauss18 the optimum is 44 and it was found 72.78, 62.85 and 58.36 using rules evolved for target graphs G18, Tree15 and G40, respectively; no rules was able to find the optimal time. For G18 the optimum is 46 and it was found 58.46, 59.08 and 61.27 using rules evolved for target graphs Gauss18, Tree15 and G40, respectively; no rules were able to find the optimal time. For G40 the optimum is 80 and it was found 97.51, 87.41 and 105.2 for the target graphs Gauss18, Tree15 and G40, respectively. Due to the arguments and results presented we concluded that reasonable results about the generalization ability of the evolved rules were found only in graphs in which an optimal solution was relatively easy to obtain.

Another kind of investigation of generalization ability of previous CA-models [4, 8] were performed in [10]. This study concentrates on the following question: if CA rule evolved based on a specific program graph has some kind of generalization ability, this rule should return reasonable results (near to the optimal) at least when it is applied to new graphs that are similar variations of the original. Simple tests were carried out to evaluate if rules evolved for Gauss18 return reasonable results when they are applied to other graphs very similar to it, using fifteen graphs with 18 tasks very similar to Gauss18. Experiments using a simple GA (without CA rules) were conducted to find reference values for the similar graphs, since their optima are unknown. Besides, 10,000 random allocations were used to calculate average makespam considering all allocations for each similar graph. Best reference values were not easy to obtain using random allocations, as we expected for graphs closely to Gauss18. Finally, the best rules evolved for Gauss18 using the model in [4] in the learning phase was reused to schedule the fifteen new graphs.

Comparing the average time found in reuse with the average time found using 10,000 random allocations, it was possible to verify evolved rules indeed perform a kind of scheduling in all the graphs, because it was possible to reduce the makespam, starting from a random initial configuration. However, if one considers the average makespam found in operation phase with the best reference value for each graph, the makespam obtained with the CA rules is around 20% above the best value, considering the fifteen graphs. Therefore, considering CA-based model described in [4, 8], rules evolved using Gauss18 as the target graph when applied to new similar graphs were able to return makespam better than random allocations. But these values are still distant from the best possible. Besides, in reference [10], an approach called *joint evolution* was proposed, which also uses more graphs besides the target one during the learning phase. However, this set of graphs is kept frozen during the evolution and no coevolutionary strategy is employed. These previous results were promising and they inspired us to propose the multigraph coevolutionary learning discussed here which returned the best results of our investigation.

5 Multigraph Coevolutionary Learning

In the new approach to the learning phase proposed here, a coevolutionary environment [2] is used to evolve CA rules able to schedule program graphs. In this model, coevolutionary interactions are modeled as predator and prey metaphor. Coevolutionary algorithms evolve different populations during the evolutionary process (one of possible solutions and another of instances to solve) and mutual feedback mechanisms between the individuals of both populations provide a strong driving force towards complexity. Two populations involved in CA-based scheduler are: P_1 formed by possible solutions to the problem (CA rules) and P_2 formed by different program graphs. P_2 is composed by the target program graph (for example, Gauss18) and its random variations, i.e., program graphs very similar to the target one. We employed a total coevolutionary model in which both populations are modified by genetic operators during the evolutionary search [9] (another model is the partial coevolutionary in which only one population is actually evolved with operators). Considering P_1, selection, crossover and mutation operators are applied in the transition rules evolution. Considering P_2, only selection and mutation are applied:

selection decides which program graph will be discarded at each generation step and mutation operator creates a new variation of the target graph.

Let T_{p1} and T_{p2} be the size of populations P_1 and P_2, respectively. P_2 is composed by the target graph and T_{p2} - 1 program graphs randomly generated based on the target one. The other individuals of P_2 are variations of the target program, each one having 1 to 4 mutations in relation to original graph. The possible modifications can be: (*i*) a different computational cost associated to a task; (*ii*) a different communication cost associated to an edge from task k to task j; (*iii*) exclusion of an edge from task k to task j; (*iv*) inclusion of a new edge from task k to task j, being the order of k lesser than the order of j and the cost associated to the new edge will be randomly generated between the biggest and the smallest communication cost of the target program.

In each generation, T_{p2} program graphs – including the target one – are used to evaluate all T_{p1} transition rules. In this evaluation, I initial configurations (IC) of the lattice are used to evaluate the ability of each CA rule of P_1 to schedule one program graph of P_2. Starting from each IC (representing an initial allocation), CA rule is applied over the lattice by M time steps and makespam T associated to the final configuration is computed. The rule fitness is given by the sum of T for all I tests.

P_1 individuals are evolved as in [4]: (*i*) randomly sorting a initial population of T_{p1} rules; (*ii*) randomly sorting a set of I initial configurations of the lattice, different for each generation and testing them on each rule as described above; (*iii*) copying the E_1 best rules (elite) without modification to the next generation. Forming the remaining T_{p1}- E_1 rules for the next generation applying the single-point crossover operator to randomly sorted pairs of elite rules, which are submitted to mutation (subject to a given rate); (*iv*) the process returns to step *ii* and continues a predefined number of generations G; (*v*) the population of the last generation is stored in a repository of rules. P_2 is also modified during the evolutionary process. The selection strategy adopted here is simple: the target graph is kept in P_2 during all execution and each program graph in P_2 participates of the evaluation of P_1 for at least 10 generations. As a consequence, the initial T_{p2} individuals are frozen in the first 10 generations. Starting from generation 11, the variation with the worst fitness in the last 10 generations is eliminated and substituted by a new one: a mutation forming by applying 1 to 4 modifications on the target graph. In operating phase, the discovered rules are applied to schedule different variations of the target graph to test the quality of discovered rules. It is expect the coevolved rules have good generalization ability.

6 Experiments

The target graph used in the experiments described here is Gauss18. We used in the operating phase 40 different variations to evaluate the rules found with coevolutionary learning. All theses variations have 18 tasks and there are 10 graphs with 2 modifications in relation to Gauss18, 10 graphs with 3 modifications, 10 graphs with 4 modifications and 10 graphs with 5 modifications.

Initially, we reproduce the environment described in [4] and explained in Section 3, where a standard GA was used to search for CA rules; we named it *GA_Learning*. A second environment was implemented using the multigraph coevolutionary learning described in last section and we named it *CO_Learning*. Both environments were implemented in C language. We choose the Selected Neighborhood because it has shown more appropriated to Gauss18 graph as presented in [4] and as observed in our own experiments. The chosen attributes for this neighborhood were the same adopted in [4]: static co-level for predecessors, computational cost for brothers and communication cost for successors. Both environments use the scheduling policy "the task with the highest dynamic level is performed first". As reported in [4] sequential updating mode has shown to be more appropriate to deal with *Gauss18*, due to its intrinsic non-linearity [5]. Some initial experiments using *GA_learning* environment confirmed it. Thus, we performed the experiments described here using sequential updating. CA and GA parameters were T_{p1} = 100, T_{p2} = 10. E_1 = 10, p_{cross} = 0.90, p_{mut} = 0.012, G = 100, I = 25 and M = 50.

The first experiment was carried out with 30 runs of *GA_learning* performing the search for CA rules able to schedule *Gauss18*. Initially, the quality of the discovered rules was evaluated in the operation phase based only on Gauss18 as in [4]. A hundred initial configurations randomly sorted were used to test each rule of the final GA population. About fifty rules were able to find an optimal scheduling (T = 44) in all the 100 ICs evaluated. The results found were compatible with [4]. Aiming to evaluate the generalization ability of these rules, we use them as schedulers in the 40 random variations of Gauss18. In these tests we also use 100 ICs to evaluate the rules for each program graph. The results are presented in Table 1, where the 40 graphs were categorized in 4 groups according to the number of variations of the graphs in respect to Gauss18: Group_2 (10 graphs with 2 modifications), Group_3 (10 graphs with 3 modifications), and so on. Table 1 presents average results of the best rule found with each environment, for each group of graphs. $T_{average}$ is the average makespam obtained using the best evolved rule to schedule the respective graph starting from 100 different IC; T_{min} is the minimum value of T found starting from 100 different IC. The table presents the average values of these metrics associated to each set of 10 graphs: $\bar{T}_{\text{average}}$ and $\bar{T}_{\text{min}}$. It also presents the average values considering the 40 graphs as a whole. Using these values we calculated the confidence interval associated to *GA_Learning*: we have 95% of confidence that average makespam is between 52.25 and 53.63.

Table 1. Operation Phase: results found aplying rules evolved in learning phase for each group of variations of Gauss18

GRAPH	GA_Learning		CO_Learning	
	$\bar{T}_{\text{average}}$	$\bar{T}_{\text{min}}$	$\bar{T}_{\text{average}}$	$\bar{T}_{\text{min}}$
Group_2	51,75	50,4	47,96	45,4
Group_3	51,03	48,2	48,02	45,5
Group_4	54,32	49,6	50,29	46,6
Group_5	54,69	51	51,15	47,1
Average	**52,94**	**49,8**	**49,35**	**46,15**
Conf. Int.	**(52,25 to 53,63)**		**(48,49 to 50,10)**	

Subsequently, a new experiment was conducted using the new coevolutionary approach presented in section 5. It was named CO_Learning and it was also formed by 30 runs. The operation phase was also tested to evaluate the generalization ability of the discovered rules, as presented in Table 1. Comparing the performance of the rules found in each learning environment, one can see that rules found by *CO_Learning* overcame those found by *GA_Learning* in all groups of 10 graphs, both in the best makespam as in the average makespam found by the best rules. This superiority reflects in the average of *T* found considering the 40 graphs: 49.35 and 52.94, for *CO-Learning* and *GA-Learning*, respectively. The confidence interval associated to the *CO_Learning* environment was also calculated: we have 95% of confidence that makespam is between 48.49 and 50.10. Comparing with the confidence interval calculated with the rules evolved with *GA_Learning* (52.25, 53.63), one can see that the coevolutionary environment returns a better performance.

We used the null hypothesis to evaluate if the rules of the second experiment are better than the first: there are significant evidences that a coevolved rule returns lower than a single evolved rule. We have 95% of confidence that this improvement is between 2.87 and 4.31 time units. As the best makespam found for these variations are close to 44 time units (optimum for Gauss18), it represents a decrease of 10% in respect to the previous model. We understand that these results qualify the coevolved rules as more general than the single evolved rules.

7 Conclusions

Starting from the multiprocessor scheduler model proposed in [4], we investigated a new approach to the learning phase, in which a coevolutionary genetic algorithm is used to search for CA rules able to schedule tasks over a parallel architecture. The coevolutionary algorithm evolves two populations simultaneously: the first formed by CA transition rules and the second formed by a target program graph and some random graphs that are similar to the target one, generated applying simple modifications in the original graph.

An important observation that we made about the related studies in [4-8] is that just a little information about the generalization ability related to the evolved rules was available and just few examples of successful reusing of such rules were indeed verified. We realized experiments applying the rules evolved in the learning phase to graphs different from the target graph and we concluded that reasonable results were found only in graphs where an optimal solution was easy to reach.

Rules evolved by the coevolutionary method proposed here have presented better generalization ability. They returned a better performance to schedule new graphs similar to the target one. It was possible to verify such ability in experimental results. These rules outperformed the scheduling accomplished by rules obtained through the strategy used in [4], that we called single evolution.

Such generalization ability is primordial to a CA-based scheduling model. The evolved rules should be able to schedule not only the target program graph but they must have an intrinsic scheduling strategy in such a way that when they are applied to a new program, optimal or suboptimal allocations are returned without the need for a new evolution. Although CA-based schedulers have been proposed with the aim to find rules with a high level of generalization, such ability was not easy to obtain in previous work [4-7]. Using the proposed coevolutionary approach, this ability was obtained at least for program graphs similar to the target one.

The results presented in the present work focus on the generalization ability of the evolved rules in respect to schedule graphs similar to the target one. Nevertheless, the desirable skill of these evolved rules is to have generalization ability not only in respect to these graphs but also for graphs more different from the graphs used to evolve the rules. We are working in this problem now but we face the results using the similar graphs as an important step to achieve this goal.

Acknowledgments. GMBO thanks to CNPq and Fapemig financial support.

References

1. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
2. Hillis, D.: Co-evolving parasites improves simulated evolution as an optimization procedure. Physica D 42(1-3), 228–234 (1991)
3. Wolfram, S.: Universality and complexity in cellular automata. Physica D 10, 1–35 (1984)
4. Seredynski, F.: Evolving cellular automata-based algorithms for multiprocessor scheduling. In: Zomaya, A., Ercal, F. (eds.) Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences, pp. 179–207 (2001)
5. Seredynski, F., Zomaya, A.Y.: Sequential and parallel cellular automata-based scheduling algorithms. IEEE Trans. Parallel Distrib. Syst. 13(10), 1009–1023 (2002)
6. Swiecicka, A., Seredynski, F., Zomaya, A.Y.: Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support. IEEE Trans. Parallel Distrib. Syst. 17(3), 253–262 (2006)
7. Swiecicka, A., Seredynski, F.: Cellular automata approach to scheduling problem. In: Proc. of the International Conference on Parallel Computing in Electrical Engineering, PARELEC 2000, Washington, DC, USA, p. 29 (2000)
8. Seredynski, F., Swiecicka, A., Zomaya, A.Y.: Discovery of parallel scheduling algorithms in cellular automata-based systems. In: IPDPS, p. 132 (2001)
9. Paredis, J.: Coevolutionary computation. Artificial Life Journal 2(3) (1996)
10. Vidica, P.M., Oliveira, G.M.B.: Cellular Automata-Based Scheduling: A New Approach to Improve Generalization Ability of Evolved Rules. In: Proc. of Brazilian Symposium on Neural Networks, pp. 18–23 (2006)
11. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing 59(3), 381–422 (1999)
12. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer Science (2008)
13. Carneiro, M.G., Oliveira, G.M.B.: Cellular automata based model with synchronous updating for task static scheduling. In: Proc. of 17th International Workshop on Cellular Automata and Discrete Complex System, AUTOMATA 2011, pp. 263–272 (2011)
14. Carneiro, M.G., Oliveira, G.M.B.: SCAS-IS: Knowledge Extraction and Reuse in Multiprocessor Task Scheduling based on Cellular Automata. Accepted for Brazilian Symposium on Neural Networks (2012, preprint)