# Energy-Aware Scheduling of Distributed Systems Using Cellular Automata

Pragati Agrawal and Shrisha Rao
pragati.agrawal@iiitb.org, shrao@ieee.org

***Abstract*—In today's world of large distributed systems, the need for energy efficiency of individual components is complemented by the need for energy awareness of the complete system. Hence, energy-aware scheduling of tasks on systems has become very important. Our work addresses the problem of finding an energy-aware schedule for a given system which also satisfies the precedence constraints between tasks to be performed by the system. We present a method which uses cellular automata to find a near-optimal schedule for the system. The rules for cellular automata are learned using a genetic algorithm. Though the work presented in this paper is not limited to scheduling in computing environments only, the work is validated with a sample simulation on distributed computing systems, and tested with some standard program graphs.**

## I. Introduction

With increasing complexity of requirements and connectivity, distributed systems have become ubiquitous. But using such systems in an energy-efficient way still needs a lot of research. Efficiency can only be achieved by proper scheduling of required tasks over many components. But even with the simplest case of a two-component system, scheduling a parallel program is an NP-complete problem [1]. Hence obtaining of optimum schedules for systems with many components is a challenging open problem.

Many exact solution methods have been employed for scheduling (for example list scheduling, critical-path-based heuristics) but they are sequential algorithms. Some limitations of sequential scheduling algorithms are their sensitivity to scheduling parameters, lack of scalability, and determinism. Due to these limitations, they are generally not able to reach an optimum solution. Parallel scheduling methods have given a new perspective to this problem, but there is yet a lot of space for research and development [2].

Stochastic global search techniques based on heuristics have also been used for scheduling. Some of the successful heuristics include genetic algorithms, neural networks, simulated annealing, and ant colony optimization. Though such methods have often produced good results, they have the drawback of requiring large scheduling overheads. Scheduling overhead is the computational cost on the system for finding a schedule dynamically. Since a new schedule has to be designed for any new program graph while the system remains the same, some properties remain common among different schedules. Generally, the stochastic search algorithms do not try to take advantage of this fact and instead search for a schedule from scratch.

To reduce the scheduling overhead by taking advantage of this property, the use of cellular automata was proposed by Seredynski and Zomaya [3] but their method was restricted to optimize the overall time taken, not at all considering energy consumption.

Energy-aware scheduling of distributed systems is unfortunately a seldom-explored area. Liu *et al.* [4] present a method for power-aware scheduling under timing-constraints, but their system assumes only one particular machine for one class of tasks, and hence does not require parallel scheduling of tasks. Artigues *et al.* [5] apply tree searches for finding schedules under energy constraints in industrial applications. Scheduling overhead is usually not a concern in industrial scheduling problems, hence the method presented in their paper does not optimize for the scheduling overhead. Wang *et al.* [6] propose a method for energy-efficient scheduling of a unicomponent system under thermal constraints. A system could fail if its peak temperature exceeds its thermal constraints; higher temperature may also lead to higher leakage power consumption. Their method performs thermal management to minimize the energy consumption in dynamic voltage/speed scaling (DVS) components. Chan *et al.* [7] propose a technique for scheduling for weighted flow time and energy graphs, which includes the scope for rejecting some jobs for overall optimality.

Cellular automata (CA) [8], [9] are collections of cells on a grid of specified shape that evolve through a number of discrete time steps according to a set of rules based on the states of neighboring cells. Cellular automata form highly parallel and distributed systems of single, locally interacting units which are able to produce a global behavior. CA can be used to adapt the properties of real-life system. Hence schedules for real-life systems can be found with less scheduling overheads, using CA.

The methods tried to date which have used cellular automata for scheduling have only concentrated on optimizing for total execution time. The method proposed by Seredynski and Zomaya [3] uses genetic algorithms to learn the rules for CA, and proposes sequential as well as parallel update rules of an irregular CA. Swiecicka *et al.* [10] present a method by adding an artificial immune system (AIS) technique to the method proposed proposed by Seredynski and Zomaya [3]. Swiecicka *et al.* [10] use linear cellular automata in place of the irregular automata of Seredynski and Zomaya [3]. Another method which uses irregular cellular automata is proposed by Ghafarian *et al.* [11], who use ant colony optimization techniques to learn the rules of their CA.

978-1-4673-0750-5/12/$31.00 ©2012 IEEE

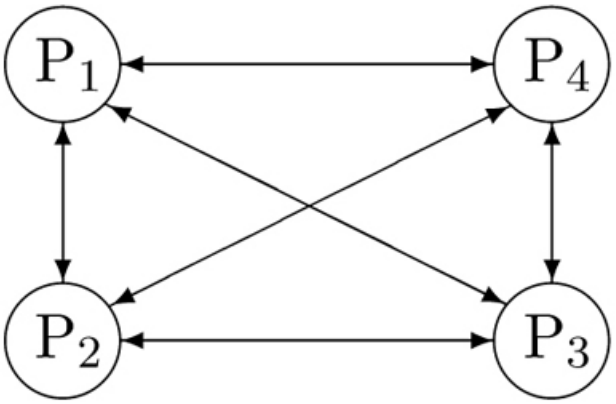


Fig. 1. System graph example: Mesh topology with four nodes

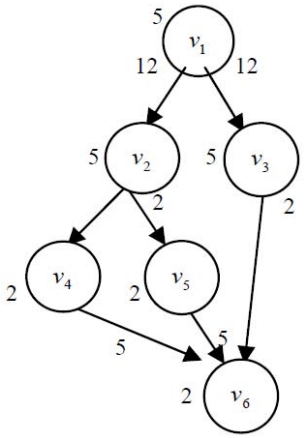


Fig. 2. Program graph example: Weights of nodes and edges are given by numbers along them. The precedence constraints are specified by direction of the edges.

This paper presents a method to try to find an optimum schedule which minimizes the total energy consumption given the system and the tasks to be completed by it. Along with the power required for task execution, the power dissipation of components when idle is also taken into consideration. Cellular automata are used for calculating good schedule. The rules of the CA are learned using genetic algorithms. A formal definition of the problem considered in this paper is presented in Section II. The proposed method is explained in Section III. Section IV illustrates the results observed by simulating the proposed method on standard program graphs. Finally, Section V describes the conclusion and future work.

## II. Energy Aware Scheduling

A distributed system is represented by an undirected unweighted graph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, called the system graph. Here,

- $\mathcal{V}_s$ is the set of nodes of the system graph representing components with their local memories. The cardinality $|\mathcal{V}_s| = N_s$ specifies the number of components in the system.
- $\mathcal{E}_s$ is the set of edges representing channels between components and defines a topology of the multicomponent system.

Figure 1 shows an example of a system graph with four nodes $P_1$, $P_2$, $P_3$ and $P_4$ as components which are connected in mesh topology with bidirectional links. It is assumed in our work that communication links themselves do not consume any power.

A parallel program is represented by a weighted directed acyclic graph $\mathcal{G}_p = (\mathcal{V}_p, \mathcal{E}_p)$, called a precedence task graph or a program graph. In the program graph:

- $\mathcal{V}_p$ is the set of nodes of the graph where each node represents an elementary task. The cardinality $|\mathcal{V}_p| = N_p$ specifies the number of elementary tasks in the program.
- $\mathcal{E}_p$ is the set of edges which specifies the precedence between the tasks.

The weight of a node represents the execution cost of the task and the weight on an edge shows the transfer cost between two tasks if they are located at different components. If they are located on the same component then the transfer cost is taken as zero. Figure 2 shows a small example of a program graph with weights of nodes and edges.

| Symbol | Meaning |
|---|---|
| $n_i$ | node number of task $i$ in program graph |
| $w(n_i)$ | weight of node $n_i$ |
| $c_{ij}$ | weight of edge connecting $n_i$ and $n_j$ |
| $P(n_i)$ | component that is assigned task $i$ |

TABLE I
Terms defined for program graph

Table I describes various parameters of a program graph.

The power consumption specifications of the system are indicated in Table II.

The total energy consumption of the system is given by:

$$E = \sum_{n=1}^{N_s} [\mu(P_n)\tau_c(P_n) + k\mu(P_n)\tau_i(P_n)] \qquad (1)$$

If the energy specifications of all system components are the same, then the minimum-energy problem reduces to that of calculating the minimum total execution time for a given program of tasks, since the minimum time corresponds to

| Symbol | Meaning |
|---|---|
| $\mu(P_m)$ | Power consumption in working state by component $P_m$ |
| $k\mu(P_m)$ | Power consumption in idle state by component $P_m$, here $0 \leq k \leq 1$ |
| $\tau_c(P_m)$ | Time taken in working state by Component $P_m$ |
| $\tau_i(P_m)$ | Time taken in idle state by Component $P_m$ |

TABLE II
Power consumption specifications of the system

minimum power. Our technique considers the general case where components have different energy requirements, hence provides an energy-aware solution. The objective is to find a schedule using cellular automata which minimizes the total energy consumption $E$ given a system graph $\mathcal{G}_s$ and a program graph $\mathcal{G}_p$. We have solved this problem using the cellular automata model.

## III. Proposed Method for Energy Aware Scheduling

To solve the scheduling problem using CA, the system graph and program graph have to be mapped to the CA domain. An elementary task of the program is mapped to a cell in the CA space. The state of the cell specifies the component to which the task is assigned. Initially, tasks are assigned randomly to the components. Then according to the rules and the neighborhood, CA evolve sequentially to reach a state which gives a near-optimum schedule. The selection of the neighborhood and the rules is critical to finding a good solution with minimal scheduling overhead. Section III-A describes the method for selecting the neighborhood. The learning of rules and finding the good schedule is explained in Section III-B.

### A. Selection of the Neighborhood

The architecture of the CA used in the proposed method is linear and irregular, which means that the neighborhood of a cell need not necessarily consist of the geometric neighbors of the cell. This provides the scope to choose neighbors which are more relevant to the task graph. Our proposed method uses a neighborhood of size 5, which includes two parents and two children of the task, and the task itself. Some terms are defined below to explain the neighborhood selection process.

- **Entry Node** – node with no parent
- **Exit Node** – node with no child
- **AEST** – Absolute Earliest Start Time
  $AEST(n_i)$ for a node $n_i$ is recursively defined as:

$$\begin{aligned} AEST(n_i) &= 0, \quad \text{for entry node} \\ AEST(n_i) &= \max_{1\le k\le p}(AEST(n_{i_k}) + w(n_{i_k}) \\ &\quad + r(P(n_{i_k}), P(n_i))c_{i_k i}), \quad \text{otherwise} \end{aligned}$$

  where $n_i$ has $p$ parent nodes and $n_{i_k}$ is its $k^{th}$ parent node, and

$$\begin{aligned} r(i,j) &= 0, \quad \text{if} \quad i = j \\ r(i,j) &= 1, \quad \text{if} \quad i \neq j. \end{aligned}$$

- **DCPL** – Dynamic Critical Path Length

$$DCPL = \arg\max_i (AEST(n_i) + w(n_i)) \tag{2}$$

- **ALST** – Absolute Latest Start Time
  $AEST(n_i)$ for a node $n_i$ is recursively defined as:

$$\begin{aligned} ALST(n_i) &= DCPL - w(n_i), \quad \text{for exit node} \\ ALST(n_i) &= \min_{1\le k\le q}(ALST(n_{i_k}) - w(n_i) \\ &\quad - r(P(n_{i_k}), P(n_i))c_{i_k i}), \quad \text{otherwise} \end{aligned}$$

  where $n_i$ has $q$ children nodes and $n_{i_k}$ is its $k^{th}$ child node.
- $\mathbf{\Delta(n_i, n_j)}$ – dynamic level distance measure between two nodes $n_i$ and $n_j$.

$$\begin{aligned} \Delta(n_i, n_j) = AEST(n_i) - AEST(n_j) + \\ ALST(n_i) - ALST(n_j) \end{aligned} \tag{3}$$

If a node $n_i$ has more than two parents (respectively, children) then the two parents (respectively, children) which have the smallest $\Delta(n_i, n_j)$ for the node $n_i$ are selected as neighbors. If less than two parents or children exist for a node, then dummy (null) parents/children are assigned. The process of assigning dummy nodes is:

1) Two dummy parent nodes with state -1 are assigned to the entry nodes.
2) Two dummy children nodes with state -1 are assigned to the exit nodes.
3) If a node has only one parent then a dummy parent node is added with the same state as the non-dummy parent node.
4) If a node has only one parent then a dummy child node is added with the same state as the non-dummy child node.

$AEST$ and $ALST$ change with states and hence neighbors may have to be changed dynamically with the schedule. This increases the scheduling overhead but promises a more relevant neighborhood. Once the scheduling problem is mapped to the cellular automata domain, we proceed to find the optimal schedule. The optimal schedule is given by the evolved state of automata which is obtained by applying a best rule to a good initial state allocation. Since both the rule and the initial state allocation are crucial, we search for both of them. We have designed a procedure in which we keep improving the state of the CA as well as the rules simultaneously to find the near-optimum schedule. We use genetic algorithms to search a the good schedule. The process is explained in the next subsection.

### B. Finding Optimal Schedule using Genetic Algorithm

A genetic algorithm [12] is a heuristic for search, mimicking the natural evolution process. In a genetic algorithm, a population of strings encoding candidate solutions evolves towards better solutions using mutation and crossover functions. The evolution usually starts from a random population of individuals and happens over multiple generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations is produced, or a satisfactory fitness level is reached for the population.

We use a genetic algorithm (GA) to search for good rules and in the process of doing so we also find the near-optimal

schedule for our scheduling problem. The application of genetic algorithms to finding CA rules was first discussed by Das *et al.* [13]. Since the aim is to search for good CA update rules, they are treated as individuals, creatures, or phenotypes in our GA setting. The fitness of a rule is given by the energy efficiency of the schedules obtained by applying the rule to some initial states of automata. Since rules can be represented as number strings, the reproduction is carried out by mutation and crossover of these strings.

The architecture of the proposed scheduling method is shown in Figure 3.

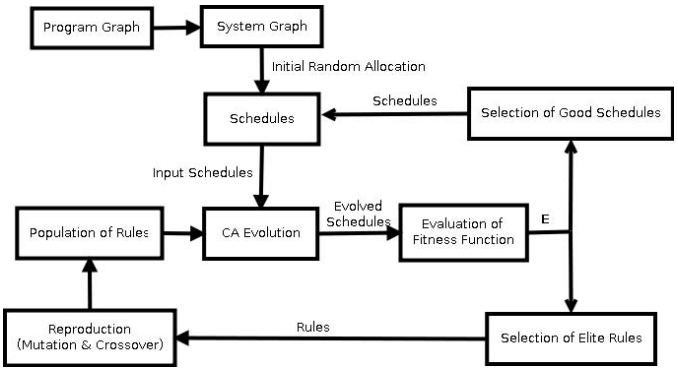


Fig. 3. Architecture of the proposed scheduling method

We will first explain the block diagram in this paragraph. The algorithm is presented later in this subsection to illustrate the process exactly.

An initial population of CA rules is randomly selected. Some example schedules are generated by randomly allocating tasks to components. Cellular automata evolve in these examples using each rule to give the evolved schedules. For each rule the average energy is calculated from these evolved schedules using (1). The best rules which give the most energy-efficient schedules are chosen for reproduction to produce the next generation of rules. These rules are also called elite rules. Mutation and crossover are applied on elite rules for reproduction to create next generation of rules. These new rules are tested by applying CA on the best schedules (allocations) derived in the previous generations and then calculating energy consumption. By iterating the above process through generations, we get the best schedule as well as the best rules are learned. The schedule is used on the program graph. The rules obtained can be saved for further use in future though they may not be optimal for a new program graph.

Our examples and simulations show how this all comes together.

## IV. Results

A number of simulations with standard program graphs have been conducted. These graphs are tree15, g40 and g18 (see, e.g., [3]). The first program graph referred to as tree15 is shown in Figure 4.

Tree15 is a binary tree with 15 nodes. All the working costs and communication costs in this program graph are the same, and can be taken to be unity.

Create an initial population $X$ (rules).
Create a set of $k$ examples (test).
**for** $q$ = 1 to $Q$ **do**
  **Begin**
  **for** $i$ = 1 to $X$ **do**
    $E_i^* = 0$
    **for** $j$ = 1 to $k$ **do**
      $E_i^* = E_i^* + CA(rule_i + test_j, CA, M_{steps})$
    **end for**
    $E_i^* = \frac{E_i^*}{k}$
  **end for**
  Sort current population of rules according to $E_i^*$.
  Move $E$(elite) best rules to the next generation.
  **for** $i$ = 1 to $X - E$ **do**
    $rule_1^{parent}$ = select()
    $rule_2^{parent}$ = select() $\neq rule_1^{parent}$
    $(rule_1^{child}, rule_2^{child})$ =
    crossover$(rule_1^{parent}, rule_2^{parent})$
    mutation$(rule_1^{child}, rule_2^{child})$
  **end for**
**end for**
Problem Solution = The best rules from $X$.

Figure 5 displays the the next program graph g40 which we considered in our simulation. It is a directed acyclic graph which has 40 nodes. The computation and communication cost of tasks are equal to 4 and 1 respectively.

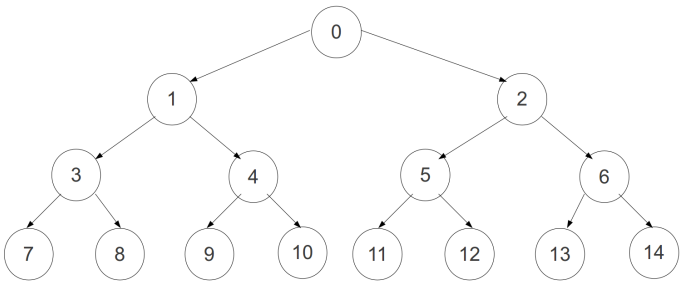


Fig. 4. Program graph tree15

The next program graph g18 is displayed in Figure 6. It has 18 tasks with different computation costs mentioned in the figure and the communication cost for all links equal to unity.

Our algorithm as well as the simulation setup allows any number of system components. All these components can have different working state power consumptions and idle state power consumptions. But most of the present scheduling algorithms consider less than 8 system components. The standard graphs which we have used are tested with less than 8 component systems in other previous approaches. Hence for illustrative purposes we have used an 8-component system, so that it is easier to compare our algorithm with others.

The other state-of-the-art scheduling algorithms compute good schedules for minimizing total time rather than energy. Though our system is energy-aware and works optimize the energy rather than time, if we take the working power and idle power as identical then it provides the schedule which minimize total time. Hence for the sake of comparison with other systems we have also calculated the schedules for total time optimality.

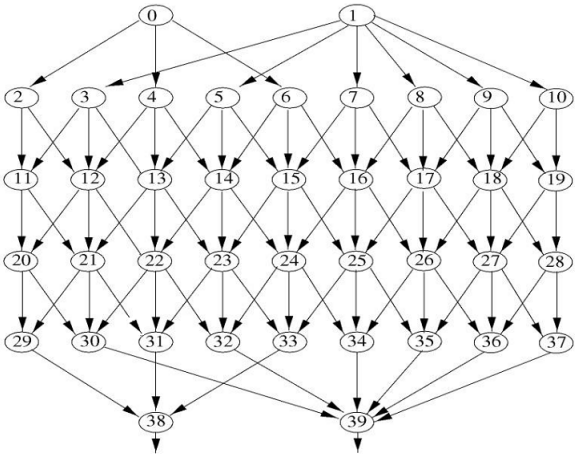


Fig. 5. Program graph g40

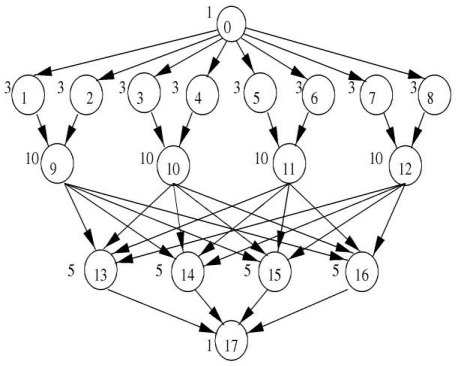


Fig. 6. Program graph g18

In the simulation reported in this Section, we assume that the cellular automata works asynchronously [3]. It means at a given instant of time, only one cell updates its state. So, a single step of CA, i.e., a rule to be applied once on all cells, will take the order of number of tasks to be completed.

In the simulations, for learning the rules we fixed the population size of GA to be 20 and the maximum number of generations to 100. First we look at the results for computation of schedules for total time optimality.

### A. Time Optimality

Scheduling algorithms with total time optimality were presented by Seredynski and Zomaya [3], Swiecicka *et al.* [10] and Ghafarian *et al.* [11]. The optimal time was computed to be 7, 24 and 33 for tree15, g18 and g40 respectively. We also achieved these times with our algorithm. For time schedules we set the working power and idle power of the system components to be alike. The progress accross different generations is shown in Table III.

| Generation | tree15 | g18 | g40 |
|---|---|---|---|
| 1 | 10 | 27.4 | 39 |
| 2 | 9.4 | 27.4 | 36.8 |
| 3 | 9.4 | 25 | 36 |
| 4 | 8.8 | 25 | 34.4 |
| 5 | 8.2 | 25.1 | 34.6 |
| 6 | 8.2 | 24.2 | 33.2 |
| 7 | 7.5 | 24.6 | 33.2 |
| 8 | 7 | 24.8 | 33 |
| 9 | 7 | 24 | 33 |
| 10 | 7 | 24 | 33 |

TABLE III
TABLE CONTAINING THE OPTIMAL TIME FOR DIFFERENT GRAPHS ACROSS GENERATIONS

### B. Energy Optimality

We have set the the working power of the components to be 1 and idle power of the components to be 0.1. Results across generations for different program graphs are shown in Figures 7, 8 and 9. These figures show the best, worst and mean values of the fitness function across generations. The weighted cumulative change for tree15 is 18.94 to 18.78, for g18 is 102.68 to 102.36, and for g40 is 173.76 to 172.32.

## V. CONCLUSION

In this paper we have presented a cellular automata based algorithm for energy-aware scheduling of tasks on a distributed system. The approach is very generic in the sense that it can be used in any distributed system. Its main strength lies in the fact that it supports any number of components and gives a schedule which is optimal according to that system's energy specifications. Generally the computational complexity for a scheduling algorithm grows geometrically with the number of tasks but this is not the case in our method. Genetic algorithms have been proved good in finding rules for CA. Our method not only finds good rules over the generations of GA, but also the optimal schedules across generations. The proposed approach opens up very promising possibilities in developing distributed energy-aware scheduling algorithms and reducing their complexity.

## REFERENCES


[1] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[2] I. Ahmad and Y.-K. Kwok, "On parallelizing multiprocessor scheduling problem," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 4, pp. 414 – 432, Apr. 1999.

[3] F. Seredynski and A. Zomaya, "Sequential and parallel cellular automata-based scheduling algorithms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 10, pp. 1009 – 1023, oct 2002.

[4] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 840 – 845.

[5] C. Artigues, P. Lopez, and A. Hait, "Scheduling under energy constraints," in *International Conference on Industrial Engineering and Systems Management (IESM09)*, 2009.

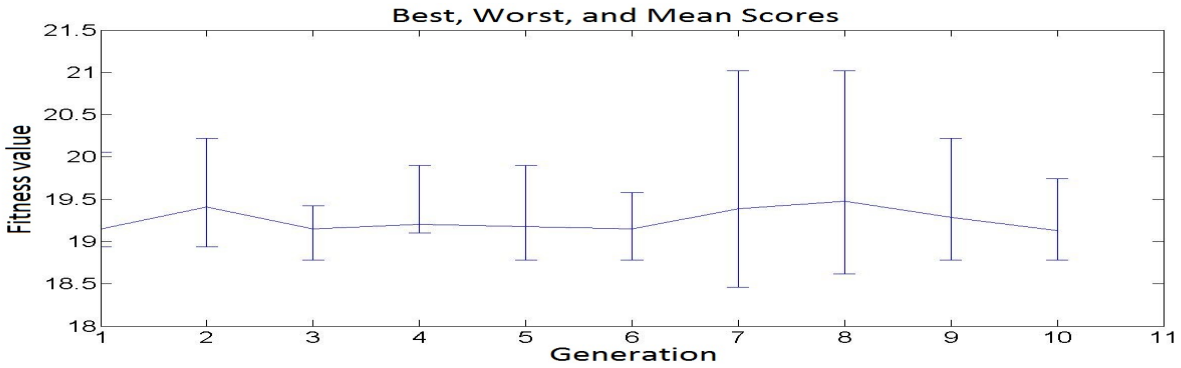


Fig. 7. Plot for graph tree15

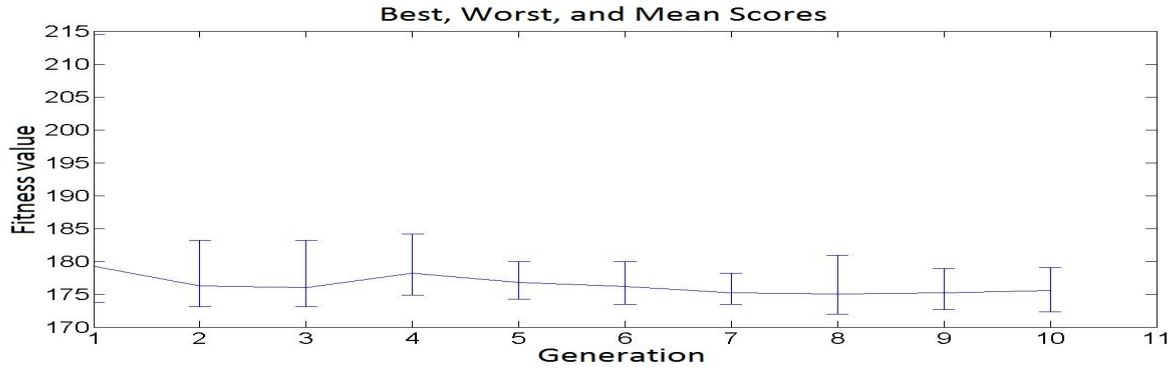


Fig. 8. Plot for program graph g40

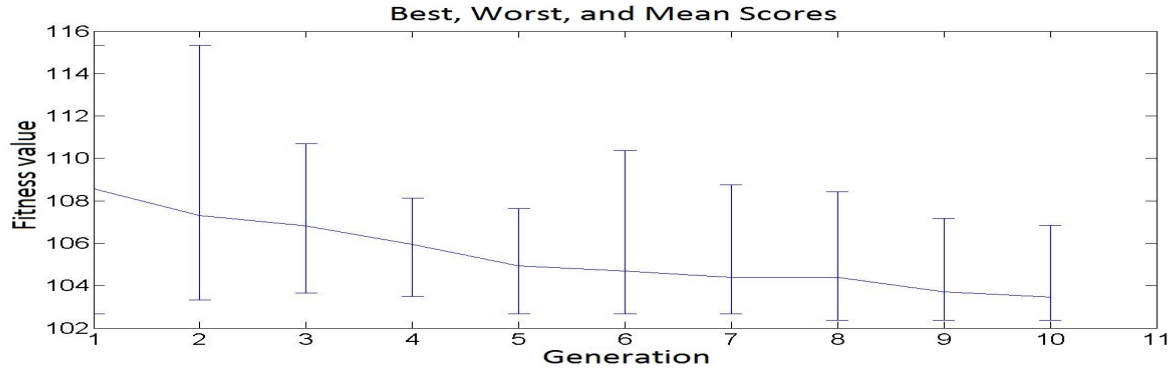


Fig. 9. Plot for program graph g18


[6] S. Wang, J.-J. Chen, Z. Shi, and L. Thiele, "Energy-efficient speed scheduling for real-time tasks under thermal constraints," in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on*, aug. 2009, pp. 201 –209.

[7] S.-H. Chan, T.-W. Lam, and L.-K. Lee, "Scheduling for Weighted Flow Time and Energy with Rejection Penalty," in *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), T. Schwentick and C. Dürr, Eds., vol. 9. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 392–403. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/3029

[8] J. L. Schiff, *Cellular Automata: A Discrete View of the World*. wiley-interscience, 2007.

[9] [Online]. Available: http://mathworld.wolfram.com/CellularAutomaton.html

[10] A. Swiecicka, F. Seredynski, and A. Zomaya, "Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 253 – 262, march 2006.

[11] T. Ghafarian, H. Deldari, and M.-R. Akbarzadeh-T, "Multiprocessor scheduling with evolving cellular automata based on ant colony optimization," in *Computer Conference, 2009. CSICC 2009. 14th International CSI*, oct. 2009, pp. 431 –436.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[13] R. Das, M. Mitchell, and J. P. Crutchfield, "A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata," in *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, ser. Lecture Notes In Computer Science, vol. 866. London, UK: Springer-Verlag, 1994, pp. 344–353.