International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009
1793-8201

A Novel Genetic Algorithm for Static Task Scheduling in Distributed Systems

Amir Masoud Rahmani and Mojtaba Rezvani

***Abstract*—The static task scheduling problem in distributed systems is very important because of optimal usage of available machines and accepted computation time for scheduling algorithm. Solving this problem using the dynamic programming and the back tracking needs much more time. Therefore, there are more attempts to solve it using the heuristic methods. In this paper, a new genetic algorithm, named TDGASA, is presented which its running time depends on the number of tasks in the scheduling problem. Then, the computation time of TDGASA to find a sub-optimal schedule is improved by Simulated Annealing (SA). The results show that the computation time of the proposed algorithm decreases compared to an existing GA-based algorithm, although, the completion time of the final scheduled task in the system decreases a little.**

***Index Terms*—Genetic algorithm, static task scheduling, distributed systems, simulated annealing, (TDGASA) Task Dependent Genetic Algorithm using Simulated Annealing**

I. Introduction

The complicated tasks can not be executed on the computing machine in an accepted interval time. Therefore, they must be divided into small sub-tasks. The sub-tasks can be executed either in the expensive multiprocessors or in the distributed systems. The latter choice is preferred due to better ratio of cost per performance. On the other hand, in most cases because of some constraints on multiprocessor systems or the natural distribution of tasks, the only optimum choice is employing the distributed systems [1].

The distributed system consists of some computing machines with different performances which are connected to each other using the high speed interconnections, and, are useful for much more computing applications [2]. The task scheduling problem in the distributed systems is known to be NP-hard, since, for allocating T tasks to H machines, the number of allocation will be $|H|^{|T|}$ and the number of states for running will be $|T|!$. One of the goals of scheduling is to determine an assignment of tasks to computing machines in order to optimize the completion time of the final task in the system.

If the number of tasks and computing systems are too high, finding the optimal or sub-optimal task scheduling would be time-consuming and in some cases it consumes more time than a random execution of tasks. Hence, we must use the heuristic algorithms based on the problem conditions instead of employing the classic methods such as the back tracking and the dynamic programming.

The heuristic algorithms prevent the popular errors in their own operation while trying to find the optimal solution. Therefore, they appear to be appropriate for solving problems [3]. There are more heuristic methods for solving the static task scheduling, some of which are: Opportunistic Load Balancing (OLB) [4], Minimum Execution Time (MET) [4], Minimum Completion Time (MCT) [4], Genetic Algorithms (GAs) [5-8], Simulated Annealing [9], Tabu Search [7].

One of the best heuristic methods is Genetic Algorithm (GA). There are many researches under the topics of solving the static task scheduling using GAs in the multiprocessor systems [3, 8, 10, 11, 12] and the distributed systems [3, 5, 6, 13, 14]. In this paper, a novel GA is presented which has a good ability to solve the above problem using the simulated annealing.

In section 2, the task scheduling problem in the distributed systems is discussed and a mathematical model is presented. The genetic algorithm and related work (Basic GA) are introduced in section 3. In section 4, the proposed algorithm is introduced. The simulation result and the comparison between the algorithms are presented in section 5 and section 6 concludes on our findings.

II. Modeling of the Scheduling Problem

A program can be considered as a set of tasks and can be modeled as a Weighted Directed Acyclic Graph as below: WDAG = *(T, <, E, D)* [13], where $T = \{t_i ; i = 1,...,n\}$ is a set of tasks, < is a partial order defined on T which specifies operational precedence constraints. That is, $t_i < t_j$ means that t_i must be completed before t_j can start execution. E is a set of directed edges. A directed edge, (i, j), between two tasks t_i and t_j specifies a partial order. D is an $n \times n$ matrix of communication data, where $D_{i,j}$ is the amount of data required to be transmitted from task t_i to task t_j.

If the distributed system consists a set of m machines which are connected to each other using a fast interconnection network, then, Estimated Completion Time (ECT) would be a $n \times m$ matrix, where $ECT_{i,j}$ shows the estimated completion time of the task t_i on the machine m_j.

A WDAG is shown in figure (1-a) and a distributed system consisting three machines is shown in figure (1-b). Table (1) illustrates the ECT matrix of the graph shown in figure (1).

DOI: 10.7763/IJCTE.2009.V1.1

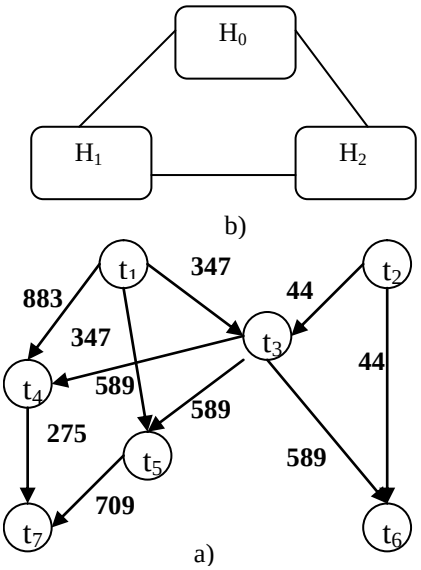

Fig. 1 a) A WDAG. b) A distributed system consisting three machines

TABLE I: THE ECT MATRIX OF THE GRAPH SHOWN FIGURE (1)

Machines / Tasks	H0	H1	H3
t_1	872	898	708
t_2	251	624	778
t_3	542	786	23
t_4	40	737	257
t_5	742	247	535
t_6	970	749	776
t_7	457	451	105

R is a $m \times m$ matrix which shows the data transfer rate between different machines. If two tasks schedule on the same machine, the communication cost of transferring data will be zero; otherwise, it is obtained based on Equation (1).

$$CommCost(t_i, t_j) = \frac{D_{i,j}}{R[H(i), H(j)]} \quad (1)$$

$D_{i,j}$ is the amount of data required to be transmitted from task t_i to task t_j and $R[H(i), H(j)]$ is the data transfer rate of two different machines.

According to the outlined concepts, the static task scheduling problem in the distributed system becomes a $\Pi: T \rightarrow H$ mapping. This mapping allocates a set of tasks T to a set of machines H, where the precedence constraints on the tasks is satisfied and the completion time of tasks on the machines is minimized. The problem's answer or Scheduling Length (SL) will be given by Equation (2).

$$Answer = \min(SL = \max\{F_j \mid j = 0, ..., m-1\}) \quad (2)$$

F_j is the completion time of final scheduled task on machine H_j including computation time, communication time and waiting time because of precedence constraints.

Two other parameters are defined for each node (task) in the graph known as b-level (the bottom level) and t-level (the top level). The b-level of a node is the length of the longest path from the node to a leaf node. If a node has no children, its b-level is equal to the average execution time of the task on the different computing machines. The t-level of a node (task) is the length of the longest path from the node to a root node in the WDAG without considering the execution time of that task. In effect, the t-level determines the earliest beginning time of a task. Therefore, if a task has no parent its t-level will be zero. Table (2) shows the average Estimated Completion Time of the tasks (AvgECT) on the different machines, the b-levels and t-levels of the graph that are shown in figure (1).

TABLE II: THE B-LEVELS AND T-LEVELS OF THE GRAPH SHOWN IN FIGURE (1)

Parameters / Tasks	AvgECT	b_level	t_level
t_1	826, 0	3767, 0	0, 0
t_2	551, 0	3189, 0	0, 0
t_3	450, 33	2594, 0	1173, 0
t_4	344, 67	957, 33	2212, 33
t_5	508, 0	1554, 67	2212, 33
t_6	831, 67	831, 67	2212, 33
t_7	337, 67	337, 67	3429, 33

III. THE GENETIC ALGORITHMS

The GAs are random searching methods based on the evolution selection and the natural phenomena. These algorithms are started with a set of random solution called initial population. Each member of this population is called a chromosome. Each chromosome is a problem solution which consists of the string genes. The number of genes and their values in each chromosome depend on the problem specification. In algorithms discussed in this paper, the number of genes of each chromosome is equal to the number of the nodes (tasks) in the WDAG and the gene values demonstrate the scheduling priority of the related task to the node (each chromosome shows a scheduling), where the higher priority means that task must be executed early.

A set of chromosomes in each iteration of GA is called a generation. The chromosomes are evaluated by their fitness functions. The offspring (the new generation) is created by applying some operators on the current generation. These operators are a) crossover which selects two chromosomes of the current population, combines them and generates a new offspring, and, b) mutation which changes randomly some gene values of a chromosome and creates a new offspring. Then, the best children and maybe their parents are selected by evolutionary select operator according to their fitness value(s).

Three phases of the production, evaluation and selection are repeated until some condition is satisfied. Finally, a chromosome which has the best fitness value(s) is selected as a solution.

A. Base Genetic Algorithm (BGA)

The GA presented by Dhodhi et. al [13], named here Base Genetic Algorithm (BGA), has four steps as shown in figure (2).

In the first step, the following parameters are read from a database: WDAG, ECT and R. Other parameters such as the initial population size (N_p), the number of generations (N_g), the crossover probability (X_r), and the mutation probability (M_r) are provided by the user.

In the second step, the b-level and t-level of each node in the WDAG are calculated and then, in the first chromosome

of the initial population, the gene value (priority) of each task is set to its b-level. The genes values (priorities) of the rest of the chromosomes are the total of the genes values of the first chromosome and the random numbers which are generated in the range of (t-level/2, -t-level/2) of the tasks.

Using the Earliest Finish Time (EFT decoding, which schedules a candidate task onto a machine on which the finish time of the task is the earliest), the overall completion time of the final task in the system is chosen as a fitness function of each chromosome. For this reason, first, all the genes (tasks) of each chromosome are sorted in descending order according to their values (priorities) in a ready queue. If the precedence of the tasks in the WDAG are not observed in a chromosome, it is chosen as an illegal chromosome and its fitness value would be infinite. Otherwise, the tasks are selected from the ready queue according to their priorities, and scheduled to the most suitable machine on which the finish time of the task is the earliest. Finally, the best chromosome of the first population is stored as a first element of the Best_Schedule array. The length of this array is equal to N_g.

Step 1. Read the WDAG, ECT, and R from a database and get N_p , N_g , X_r and M_r from the user;

Step 2. Calculate the b-level and the t-level of each task in the WDAG;
Generate Initial Population ($P_{initial}$);
$P_{current} \leftarrow P_{initial}$;
$Schedules \leftarrow Decoding\_heuristic(P_{current})$;
$Best\_Schedule \leftarrow$ evaluate ($Schedules$);

Step 3. while stop criterion not satisfied do begin
$P_{new} \leftarrow \{\}$; /* empty new population */
3-1. repeat for $(N_p/2)$ times
$dad \leftarrow select(P_{current}, Sum\_of\_fitness)$;
$mom \leftarrow select(P_{current}, Sum\_of\_fitness)$;
$P_{new} \leftarrow P_{new} \cup crsovr(dad, mom, child1, child2, X_r)$;
endrepeat;
3-2. for each chromosome $\in P_{new}$ do begin
mutate (chromosome, M_r);
endfor;
3-3.
$P_{new} \leftarrow P_{new} \cup \{four\ best\ chromosomes\ of\ P_{current}\}$
$P_{current} \leftarrow P_{new}$;
$Schedules \leftarrow Decoding\_heuristic(P_{current})$;
$Best\_Schedule \leftarrow$ evaluate ($Schedules$);
endwhile;

Step 4. Report the best schedule.

Fig. 2 The Base Genetic Algorithm (BGA) [13]

In the third step, at long as the stop criterion, i.e. the number of generations, is not satisfied, the while loop will be repeated. The third step consists of three sub-steps. In the first sub-step, the two parent chromosomes (dad and mom) are selected by the select function using a roulette wheel selection in a loop. A roulette wheel places all chromosomes in their population which every of them has its place big according to its fitness function. The chromosomes with better fitness values will be selected more often and have a higher probability of generating the offspring.

Then, the crossover operator is applied to the two parents. A random number between 0 and 1 is generated, if this number is equal or less than X_r, the parents will be selected directly for the new generation; otherwise, two children are created by the parents as it will be discussed later. For each gene a random number between 0 and 1 is generated. If that is less than 0.5 then, the gene related to the dad chromosome is copied to the first child and the gene related to the mom chromosome is copied to the second one. Otherwise, the gene related to the dad is copied to the second child and the gene related to the mom is copied to the first one. The first sub-step is repeated (N_p /2) times since in each iteration two parent chromosomes must be selected.

In the second sub-step, the mutation operator is used to prevent falling all solutions in population into the local minima of the solved problem and also used for finding the new points in the search space so that population diversity can be maintained. This operator acts on the chromosomes which are produced by crossover operator. A gene of a chromosome with a probability M_r is chosen by random and its value is added to a random number between the *-t_level/2* and *t_level/2* of that node (chromosome) in WDAG. After applying mutation, if the gene value is bigger than (*b-level+t-level*) of that node, then its value becomes (*b-level+t-level*). Also, if the gene value is less than b-level, then its value becomes b-level of that node.

Following the elitism method, in the third sub-step, the four best chromosomes with the best fitness functions are copied to the new generation. This means at least the four best solutions (schedules) are copied without changes to a new population, and therefore, the best found solution can survive until the end of run. Then, the current generation is replaced by the new one and after the decoding chromosomes using the EFT, their fitness functions are calculated and the best chromosome will be stored in the *Best_Schedule* array.

In the step four, after the Ng iteration is completed; the final stored element in the *Best_Schedule* array is the best solution for the scheduling.

IV. TDGASA: THE SUGGESTED ALGORITHM

The discussed algorithm (BGA) by Dhodhi et. al [13] has used a fixed number of generations (N_g) for each WDAG with any number of tasks. This is a problem in the above algorithm because if the number of tasks is small, there is no need to tolerate high computation time of the algorithm, and if the number of tasks in WDAG is too large, it is possible that the number of generations and the number of iterations are not enough to find an optimal or sub-optimal solution. For tackling this problem, we can use a new idea (algorithm) which its running time depends on the number of tasks.

In this new idea, two main parameters of this algorithm, i.e. N_p and N_g are defined as factors of tasks' numbers. These two factors are called *NP_factor* and *Ng_factor*.

It is obvious that if the number of tasks in a WDAG is small, the computation time will be decreased because of

lessening two above parameters. However, if the number of tasks is large the computation time of the algorithm will be increased. For decreasing the computation time, the Simulated Annealing (SA) [15] will be employed here.

To study a certain state of a material, an annealing process is used, where the material is first melted, and then slowly cooled in a controlled way to obtain a certain arrangement of the atoms. When the temperature is high, atoms can occasionally move to states with higher energy, but then, as the temperature drops, the probability of such moves is reduced. In the task scheduling algorithm, the energy of the state corresponds to its computation time, and the temperature becomes a control parameter which is reduced during the execution of the algorithm.

For decreasing the temperature and applying the geometric cooling schedule in a proper time, a new method is used. Therefore, a new algorithm, so-called Task Dependent Genetic Algorithm using Simulated Annealing (TDGASA) is introduced.

In TDGASA, if the convergence of the results is recognized after some iteration then, some parameters of the algorithm will be changed intentionally. This happens by decreasing the number of current population size (N_p) and X_r by multiplying them to a value which is less than one and also, increasing M_r by multiplying its value to a number more than one. The computation time of a new algorithm is decreased by lessening current population and the crossover probability and there is an attempt to prevent to trap with local minima by increasing the mutation probability.

Step 1. Read the WDAG, ECT and R from a database and get *Crossover_Factor, Mutation_ Factor,* N_p, N_g, N_p_Factor, N_g_Factor, *Sliding_Window,* X_r M_r, , *Population_ Factor and Comparison_ Factor,* from the user;

$N_p \leftarrow Number\_of\_tasks * N_p\_Factor$;

$N_g \leftarrow Number\_of\_tasks * N_g\_Factor$;

a)

3-4. Calculate value of CD
if (CD >= Comparison_Base)
/* So, Annealing happens */
$N_p \leftarrow N_p * Populatoin\_Factor$;
$X_r \leftarrow X_r * Crossover\_Factor$;
$M_r \leftarrow M_r * Mutation\_Factor$;
Reset Sliding_Counter to zero;
endif;

b)

Fig 3. a) The modification in the first step of algorithm shown in Figure 2. b) The fourth sub-step of step three is added to the algorithm shown in Figure 2.

TDGASA algorithm is similar to BGA in figure (2), however, the first step is modified as shown in figure (3-a) and the fourth sub-step of step three is added to the algorithm as it is shown in figure (3-b).

In the first step, in addition to reading WDAG, ECT and *R* from a database, other parameters are taken from the user.

Population_Factor, Crossover_Factor and *Mutation_Factor* are the cooling schedules of the algorithm. The suitable time for applying these factors will be explained later.

As mentioned earlier, after applying the crossover and mutation operators on the current generation and producing the new generation and also copying the four best chromosomes of the current generation without any changes to the new one, all the chromosomes of new generation are decoded and the best obtained solution is stored in the *Best_Schedule* array. Therefore, the stored fitness value (the overall completion time) of each element is better than the previous ones, so, the *Best_Schedule* is a descending array.

For applying SA method, in the fourth sub-step, the convergence of the *Best_Schedule* elements is tested by a new suggestion idea. This idea uses a sliding window which its length depends on the number of tasks in the WDAG. The Sliding Window Length (*SWL*) is given by Equation (3) as below:

$$SWL = Number\_of\_tasks * Sliding\_Window \quad (3)$$

Sliding_Window is a coefficient of the sliding window which its suitable value is determined later. If the beginning of sliding window is the index *i* of *Best_Schedule*, then the Convergence Degree (*CD*) is calculated by Equation (4):

$$CD = \frac{\sum_{k=i}^{SWL+i-1} Best\_Schedule_k}{SWL * Best\_Schedule_i} \quad (4)$$

Considering the descending array *Best_Schedule*, the *CD* is always equal or less than one. If the value of *CD* is near to one, the values of the elements are more convergent, then applying the cooling schedules will be more appropriate. The value of *CD* is compared to a *Comparison_Base*; if the *CD* is equal or more than that, the SA will happen and the sliding window counter will be set to zero; otherwise the next iteration of the algorithm will be done. The minimum distance between two SA events is equal to the *SWL*.

A. Determining the optimal parameters of TDGASA

A set of simulations is done on a Pentium IV with a 2.8 MHz Intel processor and 1 Gigabyte of RAM to determine the optimal parameters of TDGASA algorithm.

A set of 30 WDAG graphs consists of different tasks, the dependency matrix and the ECT are generated randomly by a written C# program. All elements of matrix *R* are set to one.

To find out the proper value of each parameter (*Crossover_Factor, Mutation_Factor, Population_Factor, Comparison_Base, Sliding_Window*), the different values are assigned to each parameter, while the values of the other parameters remain fixed. Then, the computation time and the overall completion time for each WDAG are calculated and accordingly, the best value for each parameter is determined as follow:

Crossover_Factor=0.6, Mutation_Factor=1.1, Population_Factor=0.8, Comparison_Base =0.95, Sliding_Window=0.25.

V. SIMULATIONS AND RESULTS

A set of simulations is done on a Pentium IV with a 2.8 MHz Intel processor and 1 Gigabyte of RAM to compare TDGASA with BGA scheduling algorithm. A set of 15 random WDAG graphs consists of 100 to 200 tasks and 4 to 9

machines are generated randomly as shown in table (3). The matrix ECT is generated by random and all elements of matrix R are set to one. X_r=0.6, M_r=0.05 are set for two algorithms and N_p=500, N_g=1000 are defined for BGA.

To run the simulations of two algorithms in the nearly same condition, the initial values for *Np_Factor* and *Ng_Factor* are set to 3 and 6 respectively, so that by multiplying their values to the number of tasks (between 100 and 200), the N_p and N_g are attained at BGA range value. The other TDGASA parameters are equal to the same values which are obtained in section 4.

TABLE III: THE WDAGS USED FOR THE COMPARISON OF TWO ALGORITHMS

Graph number	Number of tasks	Number of machines	Number of dependencies
1	104	4	563
2	117	9	42
3	124	7	1779
4	140	9	3781
5	142	7	9332
6	151	5	7768
7	155	5	6955
8	157	5	5946
9	158	8	3282
10	162	6	5416
11	179	6	676
12	180	6	4562
13	189	6	6517
14	199	5	5665
15	199	4	15205

Both of two algorithms are run for each graph three times. After scheduling, the computation time and the overall completion time (by seconds) of each algorithm are shown in table (4). Also, the average computation time of TDGASA is selected as a base and the ratio between the average computation time of BGA and the average computation time of TDGASA is calculated for each graph. Then, by adding the obtained ratios and dividing the result by the number of graphs, the average ratio is achieved. The above steps are done for the average overall completion time which is shown in table (4).

As the results show, the computation time of TDGASA is decreased by about 83 percent compared to BGA. However, the average total completion time is decreased a little.

TABLE IV: THE RESULT OF TDGASA AND BGA SCHEDULING ALGORITHMS FOR THE GRAPHS SHOWN IN TABLE III

Graph number	BGA		TDGASA	
	Average computation time (s)	Overall completion time (s)	Average computation time (s)	Overall completion time (s)
1	261	22716	75	22716
2	1576	1810	636	1772
3	402	32732	220	33263
4	684	50334	327	51186
5	935	220869	577	220869
6	1196	256129	583	256129
7	940	218869	477	210148
8	895	197929	460	197808
9	721	50808	506	51040
10	915	108393	493	109050
11	908	13629	602	13208
12	1039	84444	683	84132
13	1234	123100	891	122508
14	1279	145318	1014	151198
15	1812	553637	1638	553637
Sum of ratios	27.5282	15.0219	15	15
Average ratios	1.83520	1.0014	1	1

VI. CONCLUSIONS

The task scheduling problem in the distributed systems is known to be NP-hard. Therefore, the heuristic algorithms which obtain near-optimal solution in an acceptable interval time are preferred to the back tracking and the dynamic programming. The genetic algorithm is one of the heuristic algorithms which have the high capability to solve the complicated problems like the task scheduling.

In this paper, a new genetic algorithm, named TDGASA is presented which its population size and the number of generations depends on the number of tasks. The computation time of this algorithm is decreased by using simulated annealing. There is a tradeoff between the computation time and the total completion time. But with the proper using of simulated annealing, the computation time of the algorithm decreases more, although, the overall completion time is not increased. TDGASA proved highly influential in task scheduling problem; however, it can provide a number of contributions in fields such as the industrial engineering, the control projects, economy and etc.

REFERENCES

[1] Tanenbaum, A. S., Modern Operating Systems, Prentice Hall, 1992.
[2] Watson, D.W., Antonio, J. K., Siegel, H. Gupta, J., R., and Atallah, M.J., "Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment", Proceedings of the Heterogeneous Computing Workshop, April 1996, pp. 24–37.
[3] Haupt, R.L., Haupt, S.E., Practical genetic algorithms, John willy & Sons, 2004.
[4] Armstrong, R., Hensgen, D., and Kidd, T., "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions", 7th IEEE Heterogeneous Computing Workshop (HCW '98), 1998, pp. 79-87.
[5] Ali, S., Braun, T. D., Siegel, H. J., and Maciejewski, A. A., Heterogeneous computing, in Encyclopedia of Distributed Computing, Kluwer Academic, Norwell, MA, 2001.
[6] Braun, T. D., Siegel, H. J. and Beck, N., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed systems", Journal of Parallel and Distributed Computing Vol. 61, 2001, pp. 810-837.

[7] Zafarani Moattar E., Rahmani A.M., Feizi Derakhshi M.R., "Job Scheduling in Multi Processor Architecture Using Genetic Algorithm", 4th IEEE International conference on Innovations in Information Technology, dubai, 2007, pp. 248-251.

[8] Shenassa, M. H., Mahmoodi, M., "A novel intelligent method for task scheduling in multiprocessor systems using genetic algorithm", journal of Franklin Institute, Elsevier, 2006, pp. 1-11.

[9] Pourhaji Kazem A. A., Rahmani A. M. and Habibi Aghdam H., , "A Modified Simulated Annealing Algorithm for Static Scheduling in Grid Computing", International Conference on Computer Science and Information Technology 2008 (ICCSIT 2008), Singapore August 29 – September, 2008, pp. 623-627.

[10] Rahmani A. M., Vahedi M. A., "A Novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by Using Elitism Stepping Method", INFOCOMP – Journal of Computer Science, Vol. 7(2), 2008, pp.58-64.

[11] Abdeyazdan M. and Rahmani A. M., "Multiprocessor Task Scheduling using a new Prioritizing Genetic Algorithm based on number of Task Children", Book chapter of Distributed and Parallel Systems in Focus: Desktop Grid Computing, Springer Verlag, 2008, pp. 105-114.

[12] Lee, Y.H., Chen, C., "A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems", the 9th workshop on compiler techniques for high-performance computing, 2003.

[13] Dhodhi, M. K., Ahmad, I., Yatama, A. and Ahmad, I., "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems", Journal of Parallel and Distributed Computing, Vol. 62, 2002, pp. 1338–1361.

[14] Radulescu, A., Gemund, A.van. "Fast and effective task scheduling in heterogeneous systems", Proceeding of Heterogeneous Computing Workshop, 2000.

[15] Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi., "Optimization by simulated annealing", Science, Vol. 220, 1983, pp. 671-680.

Amir Masoud Rahmani received his B.S. in computer engineering from Amir Kabir University, Tehran, in 1996, the M.S. in computer engineering from Sharif University of technology, Tehran, in 1998 and the PhD degree in computer engineering from IAU University, Tehran, in 2005. He is an assistant professor in the Department of Computer and Mechatronics Engineering at the IAU University.
He is the author/co-author of more than 60 publications in technical journals and conferences. He served on the program committees of several national and international conferences. His research interests are in the areas of distributed systems, ad hoc and sensor wireless networks, scheduling algorithms and evolutionary computing.