Pareto Front Based Realistic Soft Real-Time Task Scheduling with Multi-objective Genetic Algorithm on Arbitrary Heterogeneous Multiprocessor System

Nafiseh Sedaghat1, Hamid Tabatabaee-Yazdi1, Mohammad-R. Akbarzadeh-T2
1Islamic Azad University, Mashhad Branch, Young Researchers Club, Iran
2Department of Electrical Engineering, Ferdowsi University of Mashhad, Iran
nf_sedaghat@yahoo.com, hamid.tabatabaee@gmail.com, akbarzadeh@ieee.org

Abstract

Task scheduling is an essential aspect of parallel processing system. This problem assumes fully connected processors and ignores contention on the communication links. However, as arbitrary processor network (APN), communication contention has a strong influence on the execution time of a parallel application. In this paper, we propose a multi-objective genetic algorithm to solve task scheduling problems with time constraints on arbitrary heterogeneous processors to find the scheduling with minimum makespan and total tardiness. To optimize objectives, we use Pareto front based technique and vector based method. In this problem, just like tasks, we schedule messages on suitable links during the minimization of the makespan and total tardiness. To find a path to transfer a message between processors we use a classic routing algorithm. We compare our method with bubble scheduling and allocation (BSA) method that is a well known algorithm. Experimental results show our method is better than BSA and yields better makespan and total tardiness.

Keywords: DAG, Edge scheduling, Genetic algorithm, Heterogeneous system, Link contention, Multi-objective optimization, Real-time system, Routing, Task scheduling.

1 Introduction

Multiprocessor heterogeneous systems are widely used in many scientific and commercial applications such as high-definition television and medical imaging [1-2]. A multiprocessor heterogeneous system involves multiple heterogeneous modules connected by arbitrary architecture that interacts with one another to solve a problem. More and more evidence show that scheduling parallel tasks is a key factor in obtaining high performance in such a system. The common objective of scheduling is to map tasks onto machines such that task precedence requirements are satisfied and minimum schedule length (makespan) are achieved [1]. In many proposed scheduling algorithms, the processors are assumed to be fully-connected, and no attention is paid to link contention or routing strategies used for communication [3]. Actually, most heterogeneous systems cannot meet this condition and their processors are linked by an arbitrary processor network (APN) [1]. Macey and Zomaya [4] showed that the consideration of link contention is significant for producing accurate and efficient schedules.

We are interested in APN algorithms that both schedule tasks and messages on arbitrary networks consisting of heterogeneous processors and communication links. Scheduling tasks while considering link contention for a heterogeneous system is a relatively less explored research topic and few algorithms for this problem have been proposed [3]; also real-time systems is not explored until now. Real-time systems are characterized by computational activities with timing constraints and classified into two categories: hard real-time system and soft real-time system. In hard real-time system, the violation of timing constraints of a certain task should not be acceptable such as patient monitoring systems and nuclear plant control. On the other hand, in soft real-time systems such as telephone switching system, image processing, usefulness of results produced by a task decreases over time after the deadline expires without causing any damage to the controlled environment [5].

All research in this topic are concerned with the minimization of a single criterion the makespan in non real-time system. However, in practice, many industries such as aircraft, electronics and semiconductors manufacturing have tradeoffs in their scheduling problems where multiple objectives need to be considered in order to optimize the overall performance of the system. Obviously, the multi-objective scheduling problems are more complex than the scheduling problems with one criterion, and it is hard to find a compromise solution because the objectives are often inconsistent, conflicting or even contradictory [6].

In this paper, we propose a genetic algorithm for soft real-time scheduling and mapping dependent tasks to an arbitrary network of heterogeneous processors. We consider link contention and just like tasks, messages are also scheduled and mapped to suitable links during the minimization of the finish time of tasks and total tardiness. To find a path for transferring a message between processors we use a traditional routing algorithm, Folyd algorithm [7].

*Corresponding author: Nafiseh Sedaghat; E-mail: nf_sedaghat@yahoo.com
To evaluate our method, we generate random DAGs with different Sparsity based on the Bernoulli distribution and compare our method with bubble scheduling and allocation (BSA) method [8].

The rest of the paper is organized as follows: Section 2 presents definition of task scheduling problem in classic and contention awareness model. In Section 3, we review some related works and in Section 4 we present our method. In Section 5, the simulation experimental results are presented and analyzed. This paper concludes with Section 6.

2 Task Scheduling

In task scheduling, the program to be scheduled is represented by a task graph.

Definition 1 (Task Graph) [9]. A task graph is a directed acyclic graph (DAG) \( G = (V, E, w, c) \) representing a program \( P \) according to the graph model. The nodes in \( V \) represent the tasks of \( P \) and the edges in \( E \) represent the communications between the tasks. An edge \( e_{ij} \in E \) from node \( n_i \) to \( n_j, i, j \in V \), represents the communication from node \( n_i \) to node \( n_j \). The positive weight \( w(n) \) associated with node \( n \in V \) represents its computation cost and the nonnegative weight \( c(e_{ij}) \) associated with edge \( e_{ij} \in E \) represents its communication cost.

The nodes are strict with respect to both their inputs and their outputs; a node cannot begin execution until all its inputs have arrived and no output is available until the computation has finished. At that time all outputs are available for communication simultaneously [10].

A schedule of a DAG is the association of a start time and a processor with every node of the DAG. To describe a schedule \( S \) of a DAG \( G = (V, E, w, c) \) on a target system consisting of a set \( P \) of dedicated processors, the following terms are defined: \( t_s(n, p) \) denotes the start time of node \( n \) on processor \( p \), \( w(n, p) \) the execution time of node \( n \) on processor \( p \). Thus, the node’s finish time is given by \( t_f(n, p) = t_s(n, p) + w(n, p) \). The processor to which node \( n \) is allocated is denoted by \( \text{proc}(n) \). For such a schedule to be feasible, the following two conditions must be fulfilled for all nodes in \( G \) [10].

Condition 1 (Processor Constraint [Dedicated Processor]). For any two nodes \( n_i, n_j \in V \),

\[
\text{proc}(n_i) = \text{proc}(n_j) = p \Rightarrow \begin{cases} t_f(n_i, p) \leq t_f(n_j, p) \\
\text{or } t_f(n_i, p) \leq t_f(n_j, p). \end{cases}
\]  

Condition 2 (Precedence Constraint [Node Strictness]). For \( n_i, n_j \in V, e_{ij} \in E, p \in P \),

\[
t_f(n_j, p) \geq t_f(e_{ij}),
\]

where \( t_f(e_{ij}) \) is the edge finish time of the communication associated with \( e_{ij} \), which is defined later, depending on the model.

Satisfaction of the first condition means that in each time only one task can be executed on a processor and satisfaction of the second condition means that every task must have received the required information from other tasks before its execution.

2.1 Classic Scheduling

Most scheduling algorithms employ a strongly idealized model of the target parallel system [5][11-13]. This model, which shall be referred to as the classic model, is defined in the following.

Definition 2 (Classic System Model). A parallel system \( M_{\text{classic}} = (P, w) \) consists of a finite set of dedicated processors \( P \) connected by a communication network. The processor heterogeneity, in terms of processing speed, is described by the execution time function \( w \). This dedicated system has the following properties:

1. local communication has zero costs,
2. communication is performed by a communication subsystem,
3. communication can be performed concurrently, and
4. the communication network is fully connected.

Based on this model, the earliest start time of a node only depends on the finish time of the node’s predecessors and the communication time between them.

Definition 3 (Earliest Start Time). The earliest start time of \( n_i \in V \), is given by

\[
t'_s(n_i) = \max_{v \in V, a \in \alpha} \begin{cases} t_f(n_i) + \\
0 \text{ if proc}(n_i) = \text{proc}(n_a) \\
\text{c(e}_{ij}) \text{ otherwise} \end{cases}
\]

Based on this system model, the edge finish time only depends on the finish time of the origin node and the communication time.

Definition 4 (Edge Finish Time). The edge finish time of \( e_{ij} \in E \) is given by

\[
t'_f(e_{ij}) = t_f(n_i) + \begin{cases} 0 \text{ if proc}(n_i) = \text{proc}(n_j) \\
\text{c(e}_{ij}) \text{ otherwise} \end{cases}
\]

Thus, communication can overlap with the computation of other nodes, an unlimited number of communications can be performed at the same time and communication has same cost \( c(e_{ij}) \). If communication is local, its cost is zero [10].
2.2 Contention Aware Scheduling

The classic scheduling model (Definition 2) does not consider any kind of contention for communication resources. To make task scheduling contention aware, and thereby more realistic, the communication network is modeled by a graph, where processors are represented by vertices and the edges reflect the communication links. The awareness for contention is achieved by edge scheduling, i.e., the scheduling of the edges of the DAG onto the links of the network graph is performed in a very similar manner to how the nodes are scheduled on the processors.

Here, it suffices to define the topology network graph to be $TG = (P, L)$, where $P$ is a set of vertices representing the processors and $L$ is a set of edges representing the communication links. Figure 3 shows an example of these systems; in this figure, weight of vertices indicates the processing speed of the processor and weight of edges indicates the speed of communication links.

The system model is then defined as follows:

**Definition 5 (Target Parallel System—Contention Model)** [10].

A target parallel system $M_{TG} = (TG, w)$ consists of a set of possibly heterogeneous processors $P$ connected by the communication network $TG = (P, L)$. This dedicated system has the following properties:

1. local communications have zero costs and
2. communication is performed by a communication subsystem.

The notions of concurrent communication and a fully connected network found in the classic model (Definition 2) are substituted by the notion of scheduling the edges $E$ on the communication links $L$. Corresponding to the scheduling of the nodes, $t_e(e, L)$ and $t_e(L)$ denote the start and finish time of an edge $e \in E$ on link $L \in L$, respectively.

When a communication, represented by the edge $e$, is performed between two distinct processors $P_{src}$ and $P_{dst}$, the routing algorithm of $TG$ returns a route from $P_{src}$ to $P_{dst}: R = \langle L_1, L_2, ..., L_i \rangle, L_i \in L$ for $i = 1, 2, ..., L$. The edge $e$ is scheduled on each link of the route. This only affects the scheduling of the nodes with an altered definition of the edge finish time (Definition 4).

**Definition 6 (Edge Finish Time — Contention Model).**

Let $G = (V, E, w, c)$ be a DAG and $M_{TG} = (TG, w)$ a parallel system. Let $R = \langle L_1, L_2, ..., L_i \rangle$ be the route for the communication of $e_j \in E, n_i, n_j \in V$, if $i$. The finish time of $e_j$ is

$$t_f(e_j) = \begin{cases} t_f(n_j) & \text{if proc}(n_i) = \text{proc}(n_j) \\ t_f(e_j, L_i) & \text{otherwise} \end{cases}$$

Thus, the edge finish time $t_f(e_j)$ in non-local communication is the finish time of $e_j$ on the last link of the route, $L_i$.

3 Related Work

There are only a few scheduling algorithms that consider arbitrary topology for processor network and contention on network links. These algorithms do not consider time constraints for tasks and discuss about non-real-time systems. This group of algorithms is called APN (arbitrary processor network) scheduling algorithms [14]. Two well-known scheduling algorithms for APNs are Dynamic-Level Scheduling (DLS) algorithm and the Bubble Scheduling and Allocation (BSA) algorithm [8].

The DLS Algorithm [8] is a list scheduling heuristic that assigns the node priorities by using an attribute called dynamic level (DL). The dynamic level of a task $t_i$ on a processor $P_j$ is equal to

$$DL(t_i, P_j) = blevel(t_i) - EST(t_i, P_j)$$

This reflects how well task $t_i$ and processor $P_j$ are matched. The blevel value of a task $t_i$ is the length of the longest path from $t_i$ to the exit task including all computation and communication costs on the path. The DLS algorithm uses static blevel value, $blevel_i$, which is computed by considering only the computation costs. At each scheduling step, the algorithm selects (ready node, available processor) a pair that maximizes the value of the dynamic level. The computation cost of tasks is set with the median values. A new term, $\Delta(t_i, P_j)$, is added to the previous equation for heterogeneous processors, which is equal to the difference between the median execution time of task $t_i$ and its execution time on processor $P_j$. The DLS algorithm requires a message routing method that is supplied by the user but no specific routing algorithm is presented in their paper.

Bubble scheduling and allocation (BSA) algorithm has two phases. In the first phase, the tasks are all scheduled to a single processor which means to effectively serialize the parallel program. Then, each task is considered in turn for possible migration to the neighbor processors. The objective of this process is to improve the finish time of tasks because a task migrates only if it can “bubble up.” If a task is selected for migration, the communication messages from its predecessors are scheduled to the communication link between the new processor and the original processor. After all the tasks in the original processor are considered, the first phase of scheduling is completed. In the second phase, the same process is repeated on one of the neighbor processors. Thus, a task migrated from the original processor to a neighbor processor may have an opportunity to migrate again to a processor one more hop away from the original processor. This incremental scheduling by migration process is repeated for all the processors in a breadth-first fashion.
4 The Proposed Algorithm

In this paper, we consider the soft real-time scheduling precedence-constrained tasks to an arbitrary network of heterogeneous processors with considering link contentions. Therefore, scheduling algorithms for such systems must schedule the tasks as well as the communication traffic by treating both the processors and communication links as equally important resources. We use genetic algorithm to find the best scheduling in this environment that has minimum finish time of tasks and total tardiness. For mapping messages on links, we use Floyd algorithm (classic routing algorithm) to determine the best path between source and destination processors. In the following, we present structure of multi-objective genetic algorithm and routing algorithm.

4.1 Multi-objective Genetic Algorithm

4.1.1 Encoding

Encoding is an effective stage in GA. Normal binary encoding does not work very well for this problem as the strings may become too long in order to incorporate all the information that is needed [15]. Therefore, the strings are encoded using decimal numbers. Each individual in the population represents a possible schedule. Figure 1 show the encoding used. Each character is a mapping between a task and processor. Each character contains the unique identification number of a task, with the encoding used. Each character is a mapping between source and destination processors. In the following, we present structure of multi-objective genetic algorithm and routing algorithm.

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 1 An Individual in GA

4.1.2 Decoding

In this stage, assignment of tasks to processors is determined so that its schedule length and its total tardiness are minimized. To do this, we schedule tasks and messages on processors and links, respectively. While the start time of a node is constrained by the data ready time of its incoming edges, the start time of an edge is restricted by the finish time of its origin node. The scheduling of an edge differs further from that of a node, in that an edge might be scheduled on more than one link. A communication between two nodes, which are scheduled on two different but not adjacent processors, utilizes the communication route of intermediate links between the two processors. The edge representing this communication must be scheduled on each of the involved links. To determine involved links we use Floyd routing algorithm that is explained in the next subsection. We use insertion policy for scheduling nodes and edges and calculate makespan and total tardiness for each individual. Makespan is calculated from this formulation:

\[
\text{makespan} = \max_{n,j} C(n_j)
\]

where \(C(n_j)\) is completion time of task \(j\).

Also total tardiness is calculated from this formulation:

\[
\text{total tardiness} = \sum_{n,j} \max \{0, (C(n_j) - D(n_j))\}
\]

where \(C(n_j)\) is completion time of task \(j\), and \(D(n_j)\) is deadline of task \(j\).

4.1.3 Evaluation Function

In each generation we need to compare individuals and sort them based on their fitness. For performing it we use a multi-objective method, vector based to determine quality of each chromosome. In this method for each solution one has to determine how many solutions dominate it and the set of solutions to which it dominates. The domination between two solutions is defined as follows [17]:

**Definition 7 (Domination).** A solution \(x^{(1)}\) is said to dominate the other solution \(x^{(2)}\), if the both following conditions are true:
1. The solution \(x^{(1)}\) is not worse than \(x^{(2)}\) in all objectives.
2. The solution \(x^{(1)}\) is strictly better than \(x^{(2)}\) in at least one objective.

All points which are not dominated by any other point are called non-dominated points of class one, or simply non-dominated points. Usually the non-dominated points together make up a front in the objective space and are often visualized to represent a non-domination front. The points lying on the non-domination front, by definition, do not get dominated by any other point in the objective space, hence they are Pareto-optimal points (together they constitute the Pareto-optimal front), and the corresponding variable vectors are called Pareto optimal solutions [17].

We calculate a non-domination set for each individual in the population and sort population based on non-domination front. The first front being completely non-dominant set in the current population and the second front being dominated by the individuals in the first front only and the front goes so on.

4.1.4 Evolution Strategy

For selection step in GA, we use tournament selection. After the selection process is completed, we use the cycle crossover method \[18\] to promote exploration as used
in [15]. For mutation, we randomly swap elements of a randomly chosen individual in the population. In each generation, the population with the current population and current offspring is sorted again based on non-domination and only the best $N$ individuals are selected (where $N$ is the population size). The initial population is generated randomly. MoGA will evolve the population until one or more stopping conditions are met. The best individual is selected after each generation and if it does not improve for 30 generations, GA stops evolving. The maximum number of generations is set at 200 because the quality of the schedules returned with more than that number does not justify the increased computation cost.

4.2 Routing Algorithm

To transfer a message from a task to another task, scheduled on different processors, we must determine a path between $P_{Src}$ and $P_{Dst}$ based on network topology graph and busy time of links for preventing contention. We propose all links are full duplex and we use store-and-forward (SAF) [8] switching for message transfer. In this paper we assume that the time of preparing a sending message on a source processor and receiving a message on a destination processor is zero. Also, we assume that the time of store-and-forward process on each intermediate processor is zero.

To find the best route for transferring message from $P_{Src}$ to $P_{Dst}$ we use Floyd algorithm [7]. First, based on speed of links, we calculate the time distances between processors and then we implement Floyd algorithm and choose the shortest path between $P_{Src}$ and $P_{Dst}$. This algorithm is simple and its results are good.

Figure 2 show an example of routing algorithm. Figure 2 (a) shows the network topology in which the weight of edges represents the speed of links between processors. Figure 2 (b) shows the network topology of which the weight of edges represents time distances between processors calculated by:

$$TD(Src, Dest) = \frac{1}{c(e_{Src, Dest})} \quad (9)$$

Where $c(e_{Src, Dest})$ is the weight of link between $Src$ and $Dest$ processor. Figure 2 (c) shows shortest path and its cost (time distance) network that is found by Floyd algorithm. Due to assuming full duplex link,

$$TD(Src, Dest) = TD(Dest, Src) \quad (10)$$

and

$$Path(Src, Dest) = Path(Dest, Src) \quad (11)$$

5 Experimental Results

In this section, we describe the performance of our algorithm on random examples including task graphs (DAGs) generated using the P-Method [19] similar to [5] [12][20].

The P-Method of generating a random DAG is based on the probabilistic construction of an adjacency matrix of a task graph. For each element, when the Bernoulli trial is a success, then the element is assigned a value of one; for a failure the element is given a value of zero. The parameter $p$ can be considered to be the Sparsity of the task graph. With this method, a probability parameter of $p = 1$ creates a

![Figure 2](image-url)
totally sequential task graph and \( p = 0 \) creates an inherently parallel one.

The P-method was used to generate 5 DAGs based on parameter \( p \in \{0.1, 0.2, 0.3, 0.4, 0.5\} \). The number of tasks in each task graph was 40. The weight of each node (computation cost of each task) and edge (communication cost between tasks) in task graph was chosen randomly based on normal distribution. For generating deadline of tasks we use this formulation:

\[
D_i = t^e_{i} + \max_{p \in P} c_{i} + \alpha
\]  

Where \( D_i \) is the deadline of task \( i; t^e_{i} \) is the earliest start time of \( i \) th task which is calculated by (3). Next term is maximum computation cost of task \( i \) on processors and \( \alpha \) is a constant that shows laxity time; in this paper we set \( \alpha = 10 \).

We schedule these task graphs with 5 arbitrary heterogeneous processors.

Figure 3 shows its topology. The weights associated with each processor show processing speed of processor and weights associated with each link show transferring rate of each link. For computing the execution time each task on each processor, it is enough to divide computation cost of the task to processing speed of processor.

Characteristics of MoGA are: population size: 40, crossover probability: 0.8, mutation probability: 0.2. To evaluate our method, we compare the results of MoGA scheduling classic routing (MoGACR) with BSA method (is a well-known link contention scheduling method). Figures 4-7 show experimental results. In these figures, horizontal axis is DAG Sparsity.

![Figure 3 Arbitrary Heterogeneous Multiprocessor System](image)

Figure 3 Arbitary Heterogeneous Multiprocessor System

Figure 4 shows the scheduling length (makespan) versus DAG Sparsity. Makespan is the first objective that we minimize it with MoGA. As shown in Figure 4, MoGACR has makespan lower than BSA and it is desirable. When the Sparsity is increased, the makespan of the two methods are closed together. This is because the sequentiality of a task graph is high and it is less difficult to find best found schedules. Figure 5 shows the total tardiness versus DAG Sparsity. Total tardiness is the second objective that we minimize. As shown in this figure total tardiness in MoGACR in sparsity = 0.1 is less than BSA method and in another DAGs total tardiness is equal to zero that show all time constraints have been satisfied.

![Figure 4 Makespan](image)

Figure 4 Makespan

![Figure 5 Total Tardiness](image)

Figure 5 Total Tardiness

Figure 6 shows the total communication time in scheduling. Communication time is the duration time of sending messages on links and it is desirable to minimize it by choosing the best route between processors. As shown in this figure, BSA has most total communication time and wastes most time for communicating. MoGACR has lower communication time than BSA due to in MoGACR we transfer messages via shortest path that is determined by Floyd algorithm.

![Figure 6 Total Communication Time](image)
The speedup is defined as the ratio of sequential time
$$\text{seq}(G) = \sum_{e \in G} w(e)$$ (local communication has zero costs)
to the makespan of the produced scheduling; therefore we
want to increase this ratio.

As shown in Figure 7, BSA has lower speedup than that
of MoGACR. When the Sparsity is increased, speedup is
decreased. This is because the sequentially of a task graph
is high and the best solution trends to sequential execution.

Figure 8 shows the Pareto solution of makespan versus
total tardiness. We expect this curve to be close to the
center. It means that we expect to have zero total tardiness
and also minimum makespan.

## 6 Conclusion

In this paper, we use multi-objective genetic algorithm
to solve the problem of soft real-time scheduling
precedence-constrained tasks to an arbitrary network
of heterogeneous processors for finding the scheduling
with minimum finish time and total tardiness. Just like
tasks, we schedule messages on suitable links during the
minimization of the finish time of tasks and total tardiness.
For finding a path for transferring a message between
processors we use classic routing algorithm known as
Floyd algorithm. In routing we select paths based on speed
of links. This problem has a large search space and because
of it we must choose the best optimization method. Due to
the fact that we have two objectives, we use vector-based
method for comparison between chromosomes. With this
method we yield better scheduling. In order to evaluate our
method, we generate random DAGs with different Sparsity
based on Bernoulli distribution. We compare results of our
method with that of BSA method. Experimental results
show our method (MoGACR) finds scheduling with lower
makespan and total tardiness than BSA in all DAGs with
different Sparsity. This means our method has capability of
facing with different Sparsity DAGs.

## References

[1] Xiaoyong Tang, Kenli Li and Padua Divid,
Communication Contention in APN List Scheduling
Algorithm, Science in China Series F: Information

and Jian-Ping Yu, On Centralized Scheduling and
Channel Assignment Scheme for WiMAX Mesh
Networks, Journal of Internet Technology, Vol.10,

[3] Yu-Kwong Kwok and Ishfaq Ahmad,
Link
Contention-Constrained Scheduling and Mapping of
Tasks and Messages to a Network of Heterogeneous
Processors, Cluster Computing: The Journal of
Networks, Software Tools and Applications, Vol.3,
No.2, 2000, pp.113-124.

[4] Benjamin S. Macey and Albert Y. Zomaya, A
Performance Evaluation of CP List Scheduling
Heuristics for Communication Intensive Task Graphs,
Proc. of the First Merged International Parallel
Processing Symposium and Symposium on Parallel
and Distributed Processing (IPPS/SPDP 1998),


Biographies

Nafiseh Sedaghat received the BS degree in computer engineering from Ferdowsi University, Mashhad, Iran in 2007. She received MS degree in artificial intelligence from Islamic Azad University, Mashhad branch, Iran in 2010. She is currently a PhD candidate in artificial intelligence at Iran University of Science and Technology, Tehran, Iran. Her research interests are intelligent systems, parallel and distributed systems, and computer networks.

Hamid Tabatabae-Yazdi received the BS degree in software engineering from Islamic Azad University, Mashhad Branch, and MS degree in artificial intelligence from Islamic Azad University, Science and Research Branch. He is currently a PhD candidate in electrical engineering with focus on control in Ferdowsi University of Mashhad, Iran. He is faculty member of Islamic Azad University, Ghouchan Branch since 1998. His research interests include soft computing, intelligent systems and evolutionary algorithms.
Mohammad-R. Akbarzadeh-T is currently a professor of electrical engineering and computer engineering at Ferdowsi University of Mashhad. In 1999, he received his PhD on *Evolutionary Optimization and Fuzzy Control of Complex Systems* from the University of New Mexico. In 2006-2007, he completed a one year visiting scholar position at Berkeley Initiative on Soft Computing (BISC), UC Berkeley. Dr. Akbarzadeh has received several awards including the IDB Excellent Performance and Leadership Awards in 2009 and 2010, and the Outstanding Faculty Awards in 2008 and 2002. His research interests are in the areas of evolutionary algorithms, fuzzy logic and control, soft computing, multi-agent systems, complex systems, robotics, and biomedical engineering systems. He has published over 230 peer-reviewed articles in these and related research fields.