IPSJ Transactions on Mathematical Modeling and Its Applications Vol.6 No.1 1–9 (Mar. 2013)

Regular Paper

Memory-efficient Genetic Algorithm for Path Optimization in Embedded Systems

Umair F. Siddiqi[1,a)] Yoichi Shiraishi[1,b)] Sadiq M. Sait[2,c)]

Received: February 1, 2012, Revised: March 23, 2012/April 3, 2012,
Accepted: April 18, 2012

Abstract: Multi-objective path optimization is a critical operation in a large number of applications. Many applications execute on embedded systems, which use less powerful processors and limited amount of memory in order to reduce system costs and power consumption. Therefore, fast and memory-efficient algorithms are needed to solve the multi-objective path optimization problem. This paper proposes a fast and memory-efficient algorithm based on a Genetic Algorithm (GA) that can be used to solve the multi-objective path optimization problem. The proposed algorithm needs memory space approximately equal to its population size and consists of two GA operations (crossover and mutation). During each iteration, any one of the GA operations is applied to chromosomes, which can be either dominated or non-dominated. Dominated chromosomes prefer the crossover operation with a non-dominated chromosome in order to produce an offspring that has genes from both parents (dominated and non-dominated chromosomes). The mutation operation is preferred by non-dominated chromosomes. The offspring replaces its parent chromosome. The proposed algorithm is implemented using C++ and executed on an ARM-based embedded system as well as on an Intel-Celeron-M-based PC. In terms of the quality of its Pareto-optimal solutions, the algorithm is compared with Non-dominated Sorting Genetic Algorithm-II (NSGA-II) and Simulated Annealing (SA). The performance of the proposed algorithm is better than that of SA. Moreover, comparison with NSGA-II shows that at approximately equal amounts of execution time and memory usage, the performance of the proposed algorithm is 5% better than that of NSGA-II. Based on the experimental results, the proposed algorithm is suitable for implementation on embedded systems.

Keywords: multi-objective shortest path problem (MOSP), Genetic Algorithm (GA), embedded systems

1. Introduction

Path optimization (PO) is a critical operation in many applications. Navigation systems of vehicles (including electric [1], [2], [3]) and unmanned aerial vehicles, path planning in robots, and path selection in computer networks are some applications that require PO. Numerous applications execute on embedded systems that have limited memory and computational speed. Navigation systems of intelligent vehicles are examples of such applications that uses less powerful embedded systems to reduce power consumption. Intelligent vehicles include electric vehicles, hybrid vehicles, and internal combustion engine based vehicles. Various algorithms can be used for PO. Algorithms that require a large amount of memory and/or are computationally intense are unsuitable for embedded systems. Therefore, algorithms that can perform good quality PO and require lesser memory and processor speed are desired.

The PO problem generally involves two or more optimization objectives and therefore is also called a multi-objective shortest path problem (MOSP) [4]. MOSPs are NP hard problems [4], [5]. The objectives in a multi-objective optimization problem can contradict each other and no single solution is said to be optimal. Therefore, multi-objective optimization algorithms determine a set of Pareto-optimal solutions. A Pareto-optimal set contains all solutions that are not dominated by any other solution found by the algorithm.

The parameter values in a graph or network in which PO is employed can change dynamically. Therefore, considering the dynamic changes in the underlying network is also important while solving the PO problem. Some examples of such dynamic changes include changes in the traffic on a road network and changes in link costs.

Evolutionary Computation (EC) algorithms have been predominantly used to solve multi-objective optimization problems. In many EC algorithms, calculations during any iteration do not depend on the results of previous iterations. Therefore, the algorithms are sufficiently robust to accommodate dynamic changes in the underlying network.

When population-based algorithms are applied to a population of solutions, they simultaneously find several solutions. This approach is suitable for multi-objective optimization problems, which have several Pareto-optimal solutions. Genetic Algorithms (GAs) [6] and Particle Swarm Optimization (PSO) algorithms [7] are examples of population-based algorithms. GAs and their different variants are found to be efficient in solving multi-objective

[1] Department of Production Science & Technology, Gunma University, Ota, Gunma 373–0057, Japan
[2] Center for Communications and Information Technology Research, Research Institute, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia
[a)] umair@emb.cs.gunma-u.ac.jp
[b)] yoichi.siraisi@gunma-u.ac.jp
[c)] sadiq@kfupm.edu.sa

optimization problems [8].

This paper proposes a GA-based algorithm to solve MOSPs. The proposed algorithm needs memory space that is approximately equal to the size of its population. The algorithm consists of two GA operations: crossover and mutation. In each iteration, all chromosomes undergo any one GA operation. Dominated chromosomes prefer the crossover operation and non-dominated chromosomes prefer the mutation operation. The second parent in the crossover operation is always a non-dominated chromosome that has some common genes with the first parent. The offspring created after the mutation or crossover operation replaces its parent chromosome. Therefore additional memory is not required to store the offspring. Pareto-optimal chromosomes are replaced only by better chromosomes. Therefore, it is an elitist algorithm.

The proposed algorithm is implemented using C++ on two different platforms: (a) a low-power ARM9 embedded system with a 500-MHz CPU and 128-MB RAM, and (b) an Intel-Celeron-M-based PC with a 1.5-GHz processor and 512-MB-RAM. The performance of the proposed algorithm is compared with Simulated Annealing (SA) [6], [9] and Non-dominated Sorting Genetic Algorithm -II (NSGA-II) [10]. SA works on only one solution and therefore is extremely memory efficient. NSGA-II is a popular algorithm for solving multi-objective optimization problems and determining Pareto-optimal solutions. It is an elitist algorithm that works on a population of solutions. The results show that the Pareto-optimal solutions found by the proposed algorithm are better than those by SA. Comparison between the proposed algorithm and NSGA-II shows that given the same amount of execution time and memory usage, the performance of the proposed algorithm is better than that of NSGA-II. Therefore, it is more suitable that the other algorithms (NSGA-II and SA) to perform PO on embedded systems.

This paper is organized as follows: Section 2 presents relevant previous work, and Section 3 describes the problem of multi-objective PO. Section 4 presents the proposed algorithm. Section 5 presents experimental results and discussion. Finally, Section 6 concludes the paper.

2. Previous Work

This section briefly discusses some existing algorithms for solving MOSPs. In addition, some GAs and their variants that are used to solve the multi-objective optimization problem are discussed. Mandow et al. [11] presented initial results of extending the A* search algorithm to the solution of MOSPs. The new algorithm is named MOA*, which is a heuristic search algorithm used to find non-dominated solutions. The search process in MOA* is guided by heuristic functions. When the guiding heuristic does not meet a certain bounding test, MOA* becomes unreliable and cannot produce any useful solution. However, it is reliable when used with a proper set of heuristics. Tsaggouris and Zaroliagis [12] proposed an improved Fully Polynomial Time Approximation Scheme (FPTAS) algorithm for solving MOSPs. Their algorithm resembles the multi-objective Bellman-Ford algorithm. Among FTPASs, it has the best time complexity. Horoba [13] performed an analysis of a simple evolutionary algorithm that consists of a fitness function and mutation operation and found that it met the requirements of a Fully Polynomial Time Randomized Approximation Scheme and its runtime was comparable to that of Tsaggouris and Zaroliagis's algorithm [12]. The conventional FTPAS requires pre-computation of some values that change to reflect dynamic changes in the network. Therefore, the conventional FTPAS does not robustly accommodate dynamic changes and the calculation of the shortest path must be restarted several times whenever there are dynamic changes in the network. Because simple EC algorithms perform well and robustly accommodate dynamic changes in the network, they are often used to solve MOSPs.

Elitist Evolutionary Multi-objective Optimization (EMO) algorithms are the most recent EMOs used for finding Pareto-optimal sets. In elitist EMO algorithms, good solutions are preserved during iterations. The following text discusses some popular elitist EMOs.

Deb et al. [10] proposed an elitist multi-objective genetic algorithm, called "Non-dominated Sorting Genetic Algorithm II" (NSGA-II) . This algorithm has low computational complexity. During iterations, it preserves two types of solutions: (i) non-dominated solutions and (ii) solutions that are the most distinct in the population. By doing so, it maintains both quality and diversity in solutions. Experimental results show that this algorithm is very successful in finding diverse Pareto-optimal sets of multi-objective optimization problems.

Bora et al. [14] applied the greedy selection reinforcement learning technique to perform self-tuning of NSGA-II algorithm parameters. Their new algorithm is named NSGA-RL. They considered four parameters of NSGA-II: probabilities of crossover and mutation operations and distribution indexes in crossover and mutation operations. The parameter values are determined on the basis of past generations and the respective results. The algorithm maintains a three-dimensional matrix that stores the rewards of crossover and mutation operations. After each crossover or mutation operation, the three-dimensional matrix is updated at the corresponding location. Reward assignment can be performed using different types of equations; therefore, different variants of NSGA-RL can be created by changing the reward assignment equation. NSGA-RL is slower than NSGA-II; however, its results are closer to those of NSGA-II with the best possible parameter values.

Zhang et al. [15] applied a GA based on the fuzzy logic inference (FLI) rules to solve the covariance matching problem, which is generally solved using an exhaustive search. FLI rules were used to determine the values of crossover and mutation probabilities. The output of FLI is based on the current iteration count, the number of iterations in which the highest fitness value remains unchanged, and equations that use the maximum and minimum fitness values. Comparison results show that the performance of their algorithm is very close to that of an exhaustive search.

Ahn et al. [16] proposed a GA for shortest-path problems. The GA consists of crossover and mutation operations. The crossover operation is performed by finding common nodes between two chromosomes and swapping their partial portions. In the mutation operation, a node is randomly selected in the chromo-

some and the subpath from the selected node to the last node in the chromosome is changed by another randomly generated subpath. The authors mentioned that their algorithm is faster than Dijkstra's Algorithm and therefore suitable for real-time operations. The limitation of their algorithm is that they did not consider the multi-objective optimization case. Li [17] proposed a PSO-based multi-objective optimization algorithm called Non-dominated Sorting Particle Swarm Optimization (NSPSO). In NSPSO, any one of the non-dominated particles is considered the best global position and is used for calculating particles velocities. The experimental results of NSPSO show that it is competitive to NSGA-II. Kim et al. [18] proposed a multi-objective optimization algorithm that uses fast non-dominated sorting and preference-based sorting. Preference-based sorting uses FLI measures to assign preferences to non-dominated solutions. The solution having the highest evaluation value is selected as the preferred solution. The algorithm requires a large memory to store the population, offspring and solution archive of size less than the population size.

The proposed algorithm is distinct from the previous algorithms because it requires a memory space equal to its population size and does not require memory to store the offspring.

3. Problem Description

For an undirected graph, $G = (V, E)$, the vertices or nodes of the graph are contained in the set V and the edges or segments that join the nodes are contained in the set E. If the graph contains a total of N_v number of vertices and N_e number of edges, then any edge $e_i \in E$ is represented as $e_i = (n_x, n_y)$, where n_x is the starting node and n_y is the ending node of the edge e_i. e_i is associated with up to K weights, i.e., $e_i.w_1, e_i.w_2, e_i.w_3, ..., e_i.w_k$. A path between a source node, i.e., n_A, and a destination node, i.e., n_B, where $n_A, n_B \in V$, is represented as: $P = \{e_1, e_2, ..., e_m\}$ such that $P \subseteq E$, $e_1 = (n_A, n_x)$, $e_m = (n_y, n_B)$, $n_x, n_y \in V$, and $m \leq N_e$. The destination node n_B should be reachable from the source node n_A in order for the path P to exist between them.

In MOSPs, the objective function value of any path P is equal to the summation of the weights of the edges that are included in it. Therefore, MOSPs can have up to K objective functions, which are represented as $f_k(P) = \sum_{e_x \in P} e_x.w_k$, for $k = 1$ to K. The goal of the optimization can be represented as $Minimize(f_1, f_2, ..., f_k)$, i.e., minimizing each objective function value.

The solution of any multi-objective optimization problem is a set of Pareto-optimal solutions, i.e., $S_{PO} = \{P_1, P_2, ..\}$, such that any $P_i \in S_{PO}$ is a complete path from a source node (n_A) to a destination node (n_B) and is not dominated by any other solution found by the algorithm. A solution dominates another solution if it is better than the other solution in at least one objective function value and is not inferior to the other solution in any objective function value.

The quality of solutions in the Pareto-optimal set is determined by measuring the hypervolume [19], [20]. The hypervolume calculates the area occupied by the Pareto-optimal set in the solution space. It measures both quality and diversity of solutions. Therefore, higher quality Pareto-optimal sets have a greater hypervolume.

4. Proposed Algorithm

This section describes the proposed algorithm in detail. A chromosome is a complete solution or path between source (n_A) and destination (n_B) nodes. Each edge in the chromosome represents a gene. The search space consists of all possible solutions. The salient features of the proposed algorithm are as follows:

(1) It consists of a set of GA operations represented as GA_{set}. GA_{set} consists of crossover and mutation operations, i.e., $GA_{set} = \{Crossover, Mutation\}$.
(2) In each iteration, all chromosomes go through any one of the GA operation.
(3) The procedure to select the GA operation for the chromosome is based on whether the chromosome is non-dominated or dominated.
(4) The offspring created after the GA operations replace their parents. Therefore, additional memory is not required to store offsprings.

The different steps of the proposed algorithm are illustrated in **Fig. 1**. The inputs to the algorithm are the network (G), source and destination nodes (n_A and n_B), population size (M), and P_b and R_z, which are real numbers between 0 and 1 and between 0.5 and 1, respectively. The first step is the initialization in which

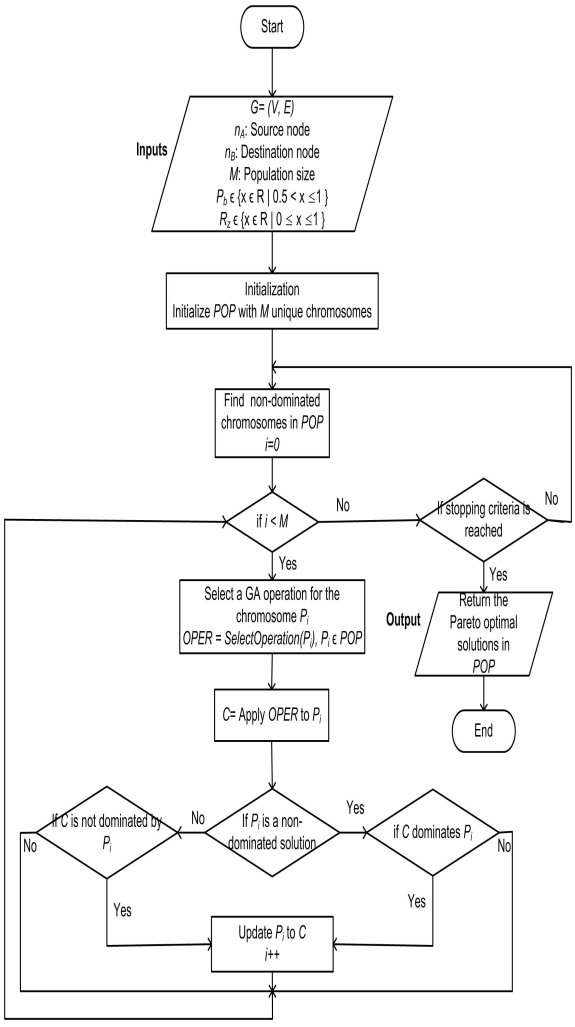

Fig. 1 Illustration of the steps of the proposed algorithm.

Input: nodes: n_A, & n_B, G=(V,E), N_e= Number of elements in E
Output: Q: Path from n_A to n_B nodes.
1: W_m[]= $random(N_e)$
2: Q= Apply Dijkstra's Algorithm (n_A, n_B)
3: **return** Q

Fig. 2 Method used to find a random path: $form\_path(n_A, n_B)$.

the population (POP) is initialized with M unique solutions. The second step is to find all non-dominated chromosomes in the population. The third step consists of a loop that executes M times. In the loop, P_i refers to a chromosome. P_i first selects a GA operation by calling the function $SelectOperation()$. The selected GA operation is stored in $OPER$, which is applied to P_i to produce a child C. The next step in the flowchart consists of three parallel decision boxes, which check the conditions under which the currently selected chromosome P_i is replaced by the newly formed child C. The stopping criterion can be the maximum time or number of iterations. After the stopping criterion is met, the non-dominated or Pareto-optimal solutions in the population are returned. The steps are described in detail in the remaining part of this section.

4.1 Initialization

The population consists of M chromosomes and is represented as $POP = \{P_0, P_1, ..., P_{M-1}\}$. Each chromosome $P_j \in POP$ is a complete path between nodes n_A and n_B. The population is initialized with M chromosomes, and each chromosome is distinct from the other chromosomes. **Figure 2** shows an algorithm used to generate a random path between nodes n_A and n_B. In line 1, the matrix W_m stores the weights of the edges. Randomly generated real numbers are stored in W_m. In line 2, Dijkstra's Algorithm is applied to find the shortest path with respect to the weights in matrix W_m. Owing to random weight assignment, the method returns a random path between nodes n_A and n_B. The method shown in Fig. 2 is also used in the mutation operation.

4.2 Finding Non-dominated Chromosomes

In each iteration, the non-dominated chromosomes in the population are marked with probability R_z in order to distinguish them from the other chromosomes. The procedure is shown in **Fig. 3**. The input is chromosomes population (POP). The chromosomes have an attribute *marked* that is set to *true* for Pareto-optimal or non-dominated chromosomes. In the second for loop, all non-dominated chromosomes are marked true with probability R_z. The $\succ$ symbol is used to indicate whether the chromosome on its left-hand side dominates the chromosome on its right-hand side.

4.3 Selection of the GA Operation

The proposed algorithm contains a set of GA operations, which is represented as GA_{set}. The set consists of two elements, i.e., $GA_{set} = \{Crossover, Mutation\}$. All chromosomes must go through any one of the operations from the set GA_{set}. The crossover operation requires that a common node should exist between the two parents [16]. In a conventional GA, parents are selected by methods such as roulette-wheel or tournament selection. This work proposes that the crossover operation can be applied to any chromosome P_j such that the second parent is any feasible pair for P_j from the population. A feasible pair for P_j is a non-dominated chromosome that has at least one common node with P_j (excluding source and destination nodes). The procedure to check if any feasible pair exists for a chromosome P_j is shown in **Fig. 4** and is represented as $CheckCn()$. The function $CheckCn(P_j)$ returns 1 if at least one pair exists for P_j in the population, otherwise it returns 0.

The method used to select a GA operation for the chromosome is shown in **Fig. 5**. The basic idea behind the proposed assignment of GA operations is the following. The chromosomes are distinguished into two classes: non-dominated and dominated. The crossover operation is selected for the dominated chromosomes with a high probability (P_b). As previously mentioned, the second parent in the crossover operation should be a non-dominated chromosome. Therefore, the dominated chromosomes have a high probability of producing a better offspring by exchanging genes with a non-dominated chromosome. On the other hand,

Input: $POP = \{P_0, P_1, ..., P_{M-1}\}$, $R_z \in \{x \in \mathbb{R} | 0 \le x \le 1\}$
Output: POP in which the Pareto-optimal chromosomes are marked
1: **for** i=0 to $M-1$ **do**
2: $P_i.marked = false$
3: **end for**
4: **for** $i = 0$ to $M-1$ **do**
5: cnt=0;
6: **for** $j = 0$ to $M-1$ **do**
7: **if** $P_j \succ P_i$ **then**
8: $cnt++$:
9: **end if**
10: **end for**
11: r: random real number between 0 and 1
12: **if** $cnt == 0$ and $r \le R_z$ **then**
13: $P_i.marked = true$
14: **end if**
15: **end for**
16: **return** POP

Fig. 3 Procedure to distinguish the Pareto-optimal chromosomes in the population.

Input: $P_j \in POP$, n_A: source node, POP: population
Output: y=1, if P_j has a feasible pair in POP. .
1: cnt=0
2: **for** each chromosome $P_i \in POP$ and $P_i \neq P_j$ **do**
3: **for** each edge $e_x = (n_a, n_b) \in P_i$ **do**
4: **for** each edge $e_y = (n_u, n_v) \in P_j$ **do**
5: **if** $n_a == n_u \neq n_A$ **then**
6: cnt++
7: Exit from the nested for loops
8: **end if**
9: **end for**
10: **end for**
11: **end for**
12: **if** $cnt > 0$ **then**
13: **return** 1
14: **else**
15: **return** 0
16: **end if**

Fig. 4 Method used to find if P_j has a feasible pair in the population POP $y = CheckCn(P_j)$.

Input: $P_j \in POP$, $P_b \in \{x \in \mathbb{R} | 0.5 < x \leq 1\}$
Output: $OPER$: A GA operation for P_j
1: v= $CheckCn(P_j)$
2: r: random real number between [0, 1]
3: **if** $v == 0$ **then**
4: **return** Mutation
5: **else if** $P_j.marked == true$ and $r \leq P_b$ **then**
6: **return** Mutation
7: **else if** $P_j.marked == true$ **then**
8: **return** Crossover
9: **else if** $r \leq P_b$ **then**
10: **return** Crossover
11: **else**
12: **return** Mutation
13: **end if**

Fig. 5 Method used to select a GA operation for the chromosome P_j, $OPER = SelectOperation(P_j)$.

a mutation operation is selected for the non-dominated chromosomes with a higher probability (P_b) so that they try to produce a better quality offspring by introducing new genes. The inputs to the method in Fig. 5 are as follows: chromosomes P_j and P_b, which is a real number between 0.5 and 1. At the end of the method, a GA operation is selected for the input chromosome P_j.

4.4 GA Operations

GA_{set} consists of two operations: crossover and mutation. This subsection describes the two operations in detail. The crossover operation in the PO problem is different from conventional single-point crossover. Before the crossover operation can be applied on the chromosome $P_j \in POP$, the second parent should be determined by calling the function $findpair(P_j)$, which is shown in **Fig. 6**. The procedure finds a feasible pair for P_j. As shown in Fig. 6, the indexes of all feasible pairs for P_j are stored in P_c. Then, an element is randomly selected from P_c and is returned. The crossover operation, which is quite similar to the one proposed by Ahn et al. [16], is shown in **Fig. 7**. The crossover operation stores the second parent in variable C. The common nodes between C and P_j are stored in C_n. Then, an element is randomly selected from C_n. In the second to last row, C is formed by combining the upper portion of C with the lower portion of P_j. C is updated to P_j on the basis of the conditions demonstrated at the start of this section with the illustration of the algorithm steps.

The proposed mutation operation is shown in **Fig. 8**. The inputs are chromosome P_j and destination node n_B. In the first three lines, three random numbers (r_1, r_2, r_3) are generated. The variable I_m stores either the edge that has maximum value of the r_1^{th} component of the weight in P_j, or a randomly selected edge (e_{r_2}) from P_j. In line 9, a new path is formed between nodes n_u and n_B. In line 10, the final offspring is created by combining the upper portion of P_j with the subpath C.

4.5 Calculation of Memory Requirement

The memory required by the proposed algorithm primarily consists of the memory that is required to store the chromosomes. Therefore, the memory requirements are determined in terms of the maximum number of chromosomes or solutions that should be stored in the memory at any time. For example, we suppose

Input: P_j, POP, n_A: source node
Output: Index of the feasible pair
1: **for** each chromosome $P_k \in POP$ **do**
2: **if** $P_k.marked == true$ and $P_j \neq P_k$ **then**
3: **for** each edge $e_x = (n_a, n_b) \in P_j$ **do**
4: **for** each edge $e_y = (n_u, n_v) \in P_k$ **do**
5: **if** $n_a == n_u \neq n_A$ **then**
6: I= index of P_k in the population POP
7: $P_c = P_c \cup I$
8: exit to the outermost for loop
9: **end if**
10: **end for**
11: **end for**
12: **end if**
13: **end for**
14: $selP$= randomly select an element from P_c
15: **return** $selP$

Fig. 6 Procedure to select a feasible pair for P_j, i.e., $y = findpair(P_j)$.

Input: P_j, POP: population of chromosomes
Output: C: offspring
1: $C = POP[findpair(P_j)]$
2: $C_n = null$
3: **for** each edge $e_x = (n_a, n_b) \in P_j$ **do**
4: **for** each edge $e_y = (n_u, n_v) \in t_1$ **do**
5: **if** $(n_a == n_u)$ **then**
6: $C_n = C_n \cup n_a$
7: **end if**
8: **end for**
9: **end for**
10: r= a randomly selected node from C_n
11: $C = concatenate(C(n_A ..., r), P_j(r, ...n_B))$
12: **return** (C)

Fig. 7 Procedure to apply the crossover operation to chromosome P_j, i.e., $C = Crossover(P_j)$.

Input: $P_j = \{e_0, e_1, ..., e_{m-1}\}$, n_B: destination node
Output: C: offspring
1: r_1: random integer between $[1, K]$
2: r_2: random integer between $[0, M-1]$
3: r_3: random integer between $[0, 1]$
4: **if** $r_3 > 0.50$ **then**
5: $I_m = (n_u, n_v) = \arg\max_{e_x \in P_j}(e_x.w_{r_1})$
6: **else**
7: $I_m = e_{r_2} = (n_u, n_v)$
8: **end if**
9: $C = form\_path(n_u, n_B)$
10: $C = concatenate(P_j(e_0, ..., e_x), C)$ (s.t. $e_x = (n_z, n_u)$)
11: **return** (C)

Fig. 8 Mutation operation, i.e., $C = mutation(P_j)$.

that a chromosome requires Δ units of memory and the proposed algorithm stores the number of chromosomes equal to the population size (M). An additional chromosome C is used in the crossover and mutation operations. Therefore, $(M + 1)\Delta$ units of memory are required to store the chromosomes. NSGA-II stores the number of chromosomes equal to twice the population size [10] because it creates the same number of children as the number of parents. The total memory requirement of NSGA-II is equal to $2N\Delta$ units. Therefore, the ratio between the memory required by NSGA-II to that required by the proposed algorithm

is equal to $\frac{mem_{NSGA-II}}{mem_{Proposed}} = \frac{2N}{M+1}$. SA preserves the current solution and one neighboring solution and therefore requires maximally 2Δ units of memory.

5. Experimental Results and Discussion

The performance of the proposed algorithm is compared with (i) SA, which works on a single solution, and (ii) NSGA-II, which is one of the most successful multi-objective optimization algorithms. The algorithms are implemented using C++ and executed on an ARM-based embedded system and Intel-Celeron-M-based PC. A Celeron-M processor is also used in the embedded system owing to its power efficiency. The ARM-based embedded system consists of a Techlogix TS-7800 embedded system [21] that has a 500-MHz ARM9 processor and 128-MB DRAM. The PC comprises a 1.5-GHz Celeron-M processor with 512-MB memory. This section presents the parameter values used in the implementation, details of the experimental setup, and a discussion about the performance of the algorithms.

5.1 Algorithms Parameters

The value of K was set to 3 because the multi-objective PO problem in the experiments has three objective functions. The multi-objective optimization problem can be represented as follows:

$Minimize(f_1(P_j), f_2(P_j), f_3(P_j))$, where P_j is a chromosome in the population.

The stopping criterion in all algorithms and test cases was set to 10 s. The proposed algorithm was implemented with the following parameter values: population size (M) was set to 10 and 20, P_b was set to 0.65, and R_z was set to 1. SA implementation has the following parameters: initial temperature (T_0) = 100, cooling rate (α) = 0.8, and constant (β) = 0.85. SA and its metropolis function, as described by Sait et al. [6], were used. The neighboring element in SA was determined using the mutation operation [16]. The cost function in SA was equal to the square root of the summation of the squares of the different objective functions. NSGA-II implementation had population sizes (N) of 5 and 10. When M = 10 and N = 5, the ratio $\frac{mem_{NSGA-II}}{mem_{Proposed}} = 0.91$. When M= 20 and N= 10, the ratio $\frac{mem_{NSGA-II}}{mem_{Proposed}} = 0.95$. Therefore, although the proposed algorithm uses a larger population than NSGA-II, the memory requirement of the two algorithms were approximately equal. Thus, we can say that the tests performed with approximately the same execution time and used the same amounts of memory. The details of the NSGA-II implementation are as follows. It used tournament selection based on crowding distance to select the parents for the crossover operation. The crossover and mutation operations were used for the PO problem, as proposed by Ahn et al. [16]. Crossover probability was set to 0.90 and mutation probability was set to 0.15. During each iteration, the elements for the population in the next iteration were selected from the population and children sets. The selection was made based on the non-domination count and diversity of solutions.

5.2 Experimental Setup

The algorithms were developed on a PC and then compiled for the ARM9 embedded system by using the C++ cross-compiler for the ARM-based embedded system [21]. The executable file was transferred from the PC to the embedded system through a serial port. The implementations of the proposed algorithm, NSGA-II and SA were represented as "proposed," "NSGA-II," and "SA", respectively. The undirected graphs were generated using a random graph generation tool [22]. **Table 1** lists the number of nodes and edges in the graphs. The graphs labeled $SG0 - SG5$ were executed on the ARM-based embedded system and those labeled $BG0 - BG5$ were executed on the Celeron-M PC. The edges can have up to three weights and their values were assigned as random real numbers between 0 and 200. During any test instance, the source and destination nodes were randomly selected in the given graph. Then, the three algorithms (proposed, NSGA-II, and SA) were executed to obtain their Pareto-optimal solutions. The stopping criterion in the algorithms, as previously mentioned, was set to 10 s. A test on any graph comprised up to 10 test instances. Tests were conducted on all graphs and repeated for different values of M and N.

Table 1 Characteristics of the graphs.

Graph	Number of nodes (N_v)	Number of edges (N_e)
$SG0$	250	1,000
$SG1$	300	1,300
$SG2$	190	670
$SG3$	240	1,250
$SG4$	190	800
$SG5$	270	1,100
$BG0$	1,900	7,700
$BG1$	1,700	7,700
$BG2$	1,800	8,400
$BG3$	2,000	8,800
$BG4$	1,700	7,600
$BG5$	2,000	9,000

The hypervolumes of the Pareto-optimal sets were calculated using the tool proposed by Fonseca et al. [23]. The tool uses an improved version of the Hypervolume by Slicing Objectives algorithm [24], which accurately computes the hypervolume. This algorithm is among the fastest methods to compute hypervolume. The hypervolume [19] indicator measures both quality and diversity of solutions. In hypervolume calculations, the bounding point was selected by the method employed by Knowles [25]. He selected the bounding point as $b_j = \max_j + \delta(\max_j - \min_j)$, where b_j is the bounding value of the j^{th} coordinate and $\max_j$ and $\min_j$ are the maximum and minimum values, respectively, of the j^{th} coordinate in the Pareto-optimal solutions. The value of δ is taken as 0.01. The hypervolume distributions can be represented using box-and-whisker charts. The Wilcoxon rank sum tests [26] were used to compare the hypervolume distributions from two different algorithms. The rank sum tests were applied at a significance level of 5% (α = 0.05).

5.3 Results

Figure 9 shows the box plots on two graphs of the hypervolumes of the Pareto-optimal solutions that were found by the different algorithms. The box plots show that the hypervolumes of the proposed algorithm and NSGA-II are closer to each other and have higher values than the hypervolumes obtained from SA. The box plots on the remaining graphs also show a similar trend. The

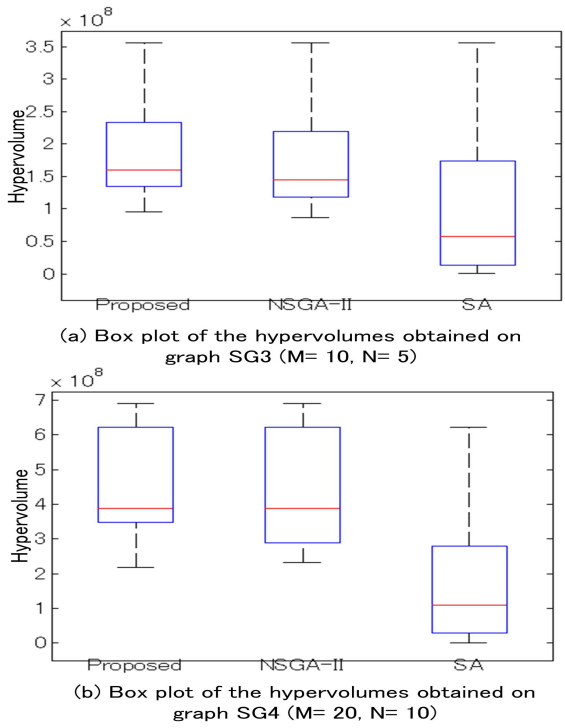

Fig. 9 Box-and-whisker plots of the hypervolumes of the Pareto-optimal sets.

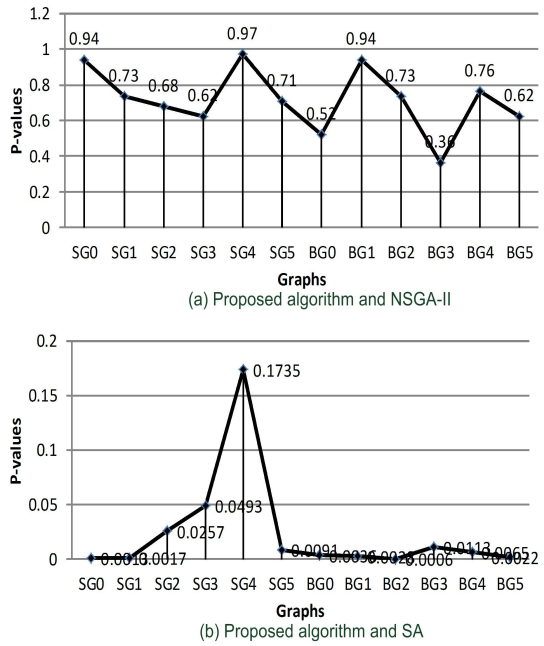

Fig. 10 Wilcoxon rank sum test results when M = 10 and N = 5.

Wilcoxon rank sum tests were applied in all tests and the results are shown in **Figs. 10** and **11**. Figure 10 shows the results when $M = 10$ and $N = 5$ and Fig. 11 shows the results when $M = 20$ and $N = 10$. The results show that the statistical difference between the hypervolume distributions obtained from the proposed algorithm and NSGA-II was insignificant. On the other hand, the P-values of the rank sum tests on the proposed algorithm and SA show that the two hypervolume distributions were statistically different from each other. Therefore, based on the box-plots and rank sum test results, we can conclude that the proposed algorithm provides Pareto-optimal solutions that are equal in quality

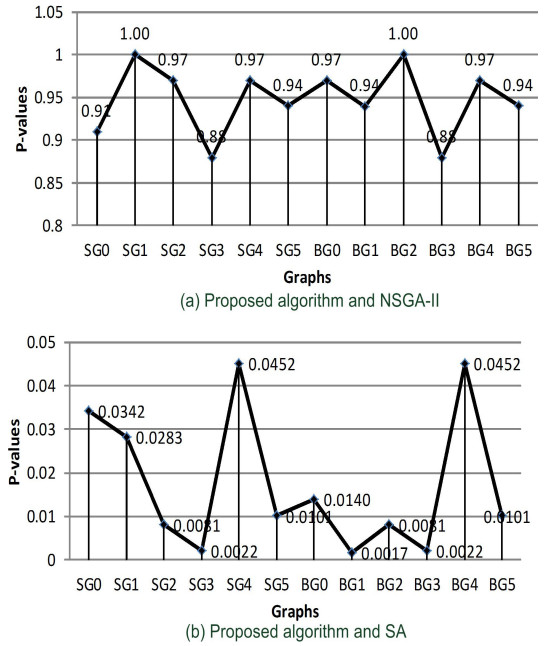

Fig. 11 Wilcoxon rank sum test results when M = 20 and N = 10.

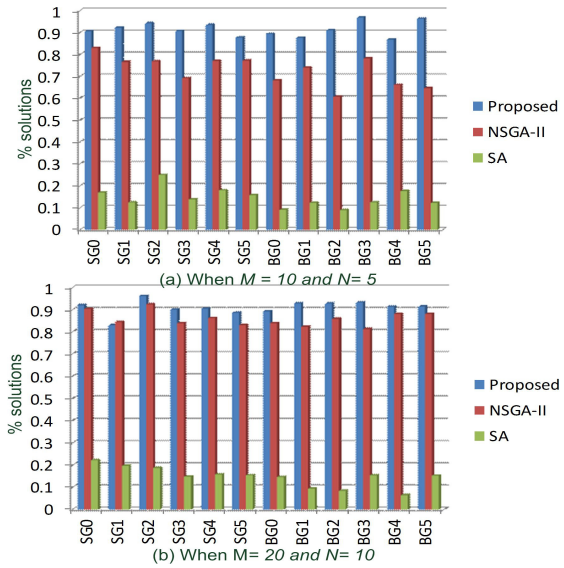

Fig. 12 Contribution of different algorithms in the overall Pareto-optimal solutions.

to those of NSGA-II and are better than those of SA.

In addition, the performance of the algorithms was measured by determining their share in the overall Pareto-optimal solutions. The algorithms that have a higher share are better, and these can be determined by the following method. During any test instance, the results of the algorithms were combined and an overall Pareto-optimal set was calculated. The number of solutions from any algorithm that also exist in the overall Pareto-optimal set determines its share in the overall Pareto-optimal set. The results are shown using bar graphs in **Fig. 12**. The x-axis indicates the network on which tests were performed, and the y-axis shows the percentage of solutions in the overall Pareto-optimal set that are contributed by different algorithms. When M = 10 and N = 5, the proposed algorithm performs better than NSGA-II and SA in most test cases. Similarly, when M = 20 and N = 10, the proposed algorithm performs better than NSGA-II and SA in most

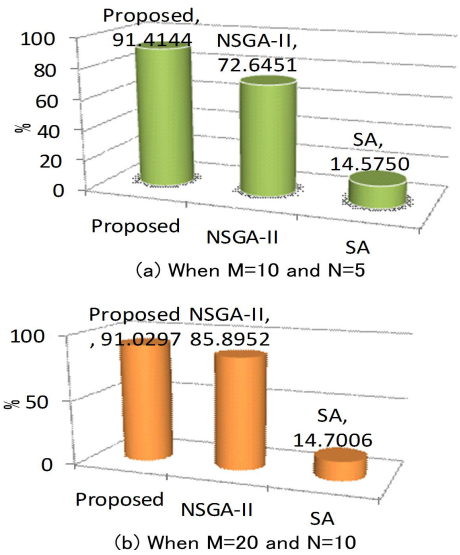

Fig. 13 Average contribution of the algorithms to the overall Pareto-optimal solutions.

test cases. The average test results are shown in **Fig. 13**, which shows that the average performance of the proposed algorithm is better than those of NSGA-II and SA. Comparison of the average contribution to the overall Pareto-optimal solutions suggests that when the execution time and memory usage of the proposed algorithm and NSGA-II are approximately equal, the performance of the proposed algorithm is better than that of NSGA-II. On the other hand, SA consumes less memory, but its performance is not competitive to that of the proposed algorithm and NSGA-II.

Therefore, based on the experimental results described in this section, the performance of the proposed algorithm is 18.8% better than that of NSGA-II when $\frac{mem_{NSGA-II}}{mem_{Proposed}} = 0.90$. Moreover, the performance of the proposed algorithm is 5.1% better than that of NSGA-II when $\frac{mem_{NSGA-II}}{mem_{Proposed}} = 0.95$.

6. Conclusions

This work proposed a memory-efficient GA-based algorithm for solving the multi-objective PO problem. The proposed algorithm has a memory requirement that is approximately equal to the size of its population. In each iteration, all chromosomes (or solutions) go through the crossover or mutation operation. The crossover operation is preferred for a dominated chromosome in the case that the other parent is a non-dominated chromosome. Using the crossover operation, dominated chromosomes can exchange some genes with non-dominated chromosomes and increase their probability of producing a better offspring. The mutation operation is preferred for non-dominated chromosomes. In this operation, a chromosome makes random changes in its genes to produce a better offspring. The offspring replaces its parent if it is not dominated by the parent. The chromosomes that are marked as Pareto-optimal are only replaced by their offspring if they dominate them.

The proposed algorithm was implemented on two different platforms: an ARM-based embedded system with a 500-MHz processor and 128-MB RAM and an Intel-Celeron-M based PC with a 1.5-GHz processor and 512-MB RAM. In addition, the proposed algorithm was compared with NSGA-II and SA. The comparison results measured the quality of the Pareto-optimal solutions obtained with the algorithms when their execution time and memory usage were approximately equal. We found that SA is the most memory-efficient, but its solution quality is not as good as the other algorithms. The performance of the proposed algorithm was 5% better than that of NSGA-II. The experimental results show that the proposed algorithm is suitable to perform multi-objective POs in embedded systems, which generally have less powerful processors and limited memory. In the future, more GA operations can be added to the set GA_{set}, which can further improve its performance.

Acknowledgments Acknowledgement are due to KDDI Foundation, Japan and King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia for their support.

References

[1] Siddiqi, U.F., Shiraishi, Y. and Sait, S.M.: Multi Constrained Route Optimization for Electric Vehicles using SimE, *International Conference on Soft Computing and Pattern Recognition* (*SoCPaR 2011*), pp.376–383 (2011).
[2] Siddiqi, U.F., Shiraishi, Y. and Sait, S.M.: Multi Constrained Route Optimization for Electric Vehicles (EVs) using Particle Swarm Optimization (PSO), *11th International Conference on Intelligent Systems Design and Applications* (*ISDA*), pp.391–396 (2011).
[3] Siddiqi, U.F., Shiraishi, Y. and Sait, S.M.: Multi Objective Optimal Path Selection in the Electric Vehicles, *Artificial Life and Robotics*, Vol.17, No.1, pp.113–122 (2012).
[4] Tarapata, Z.: Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adoption of Standard Algorithms, *Int. J. Appl. Math. Comput. Sci.*, Vol.17, No.2, pp.269–287 (2007).
[5] Garey, M.R. and Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co. (1997).
[6] Sait, S.M. and Youssef, H.: *Iterative Computer Algorithms with Applications in Engineering*, pp.50–148, IEEE Computer Society Press (1999).
[7] Kennedy, J. and Eberhart, R.: Particle Swarm Optimization, *Proc. 1995 IEEE International Conference on Neural Network*, pp.1942–1948 (1995).
[8] Konak, A., Coit, D.W. and Smith, A.E.: Multi-Objective Optimization using Genetic Algorithms: A tutorial, *Reliability Engineering & System Safety*, Vol.91, No.9, pp.992–1007 (2006).
[9] Kirkpatrick, S. Jr., Gelatt, C. and Vecchi, M.: Optimization by Simulated Annealing, Science, Vol.220, No.4598, pp.498–516 (1983).
[10] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Trans. Evolutionary Computation*, Vol.6, No.2, pp.182–197 (2002).
[11] Mandow, L. and Perez de la Cruz, J.L.: A New Approach to Multiobjective A* Search, *Proc. IJCAI'05*, pp.218–223 (2005).
[12] Tsaggouris, G. and Zaroliagis, C.: Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications, *Journal Theory of Computer Systems*, Vol.45, No.1, pp.162–186 (2009).
[13] Horoba, C.: Exploring the Runtime of an Evolutionary Algorithm for the Multi-Objective Shortest Path Problem, *Evolutionary Computation*, Vol.18, No.3, pp.357–381 (2010).
[14] Bora, T.C., Lebensztajn, L. and Coelho, L.D.S.: Non-Dominated Sorting Genetic Algorithm Based on Reinforcement Learning to Optimization of Broad-Band Reflector Antennas Satellite, *IEEE Trans. Magnetics*, Vol.48, No.2, pp.767–770 (2012).
[15] Zhang, X., Hu, S., Chen, D. and Li, X.: Fast Covariance Matching with Fuzzy Genetic Algorithm, *IEEE Trans. Industrial Informatics*, Vol.8, No.1, pp.148–157 (2012).
[16] Ahn, C.W. and Ramakrishna, R.S.: A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations, *IEEE Trans. Evolutionary Computation*, Vol.6. No.6 (2002).
[17] Li, X.: A Nondominated Sorting Particle Swarm Optimizer for Multiobjective Optimization, *International Conference on Genetic and Evolutionary Computation* (*GECCO 2003*) *Part I*, pp.37–48 (2003).
[18] Kim, J.-H., Han, J.-H., Kim, Y.-H., Choi, S.-H. and Kim, E.-S.: Preference-Based Solution Selection Algorithm for Evolutionary Multiobjective Optimization, *IEEE Trans. Evolutionary Computation*, Vol.16, No.1 (2012).

[19] Ngatchou, P., Zarei, A. and El-Sharkawi, M.A.: Pareto Multi Objective Optimization, *Proc. 13th Intelligent Systems Application to Power System*, pp.84–91 (2005).
[20] Bader, J.M.: Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods, Ph.D. dissertation, Swiss Federal Inst. Technology (ETH), Zurich, Switzerland (2009).
[21] TS-7800 Datasheet, available from ⟨http://www.embeddedarm.com/documentation/ts-7800-datasheet.pdf⟩.
[22] Viger, F. and Latapy, M.: Efficient and Simple Generation of Random Simple Connected Graphs with Prescribed Degree Sequence, *11th Conference of Computing and Combinatorics* (*COCOON 2005*), pp.440–449 (2005).
[23] Fonseca, C.M., Paquete, L. and López-Ibáñez, M.: An Improved Dimension—Sweep Algorithm for the Hypervolume Indicator, *2006 IEEE Congress on Evolutionary Computation* (*CEC'06*), pp.1157–1163 (2006).
[24] White, L., Hingston, P., Barone, L. and Husband, S.: A Faster Algorithm for Calculating Hypervolume, *IEEE Trans. Evolutionary Computation*, Vol.10, No.1, pp.29–38 (2006).
[25] Knowles, J.: ParEGO: A Hybrid Algorithm With On-Line Landscape Approximation for Expensive Multiobjective Optimization Problem, *IEEE Trans. Evolutionary Computation*, Vol.10 No.1, pp.50–66 (2005).
[26] Hollander, M. and Wolfe, D.A.: *Nonparametric Statistical Methods*, John Wiley & Sons, Inc. (1999).

Umair F. Siddiqi received his B.E. degree in Electrical Engineering from NED University of Engineering & Technology, Karachi, Pakistan in 2002 and M.S. degree in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia in 2007. He is now a Ph.D. Student in the Faculty of Engineering, Gunma University, Japan. He has published seven international conference papers and five journal papers. His research interests are: Computational Intelligence, Evolutionary Algorithms, Artificial Intelligence, and Embedded Systems. He is a student member of IEEE.

Yoichi Shiraishi received his B.S., M.S. and Ph.D. degrees in 1979, 1981 and 1993, respectively, all from Tokyo Institute of Technology. He has been engaged in the research and development of ECAD/EDA algorithms at Hitachi's Central Research Laboratory from 1981 to 1993. Now, he is with the Department of Production Science and Technology, Gunma University. His research includes combinatorial optimization algorithms, stochastic algorithms, embedded system design, signal processing, chaos theory and hardware design. He is a member of IEICE, JIEP, JFS, JSIAM, SICE, IEEE and ACM.

Sadiq M. Sait obtained a Bachelor's degree in Electronics from Bangalore University in 1981, and Master's and Ph.D. degrees in Electrical Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in 1983 & 1987 respectively. Since 1987 he has been working at the Department of Computer Engineering where he is now a Professor. In 1981 Sait received the best Electronic Engineer award from the Indian Institute of Electrical Engineers, Bangalore (where he was born). In 1990, 1994 & 1999 he was awarded the "Distinguished Researcher Award" by KFUPM. In 1988, 1989, 1990, 1995 & 2000 he was nominated by the Computer Engineering Department for the 'Best Teacher Award' which he received in 1995 and 2000. Sait has authored over 200 research papers, contributed chapters to technical books, and lectured in over 25 countries. Sadiq M. Sait is the principle author of the books (1) VLSI PHYSICAL DESIGN AUTOMATION: Theory & Practice, published by McGraw-Hill Book Co., Europe, (and also co-published by IEEE Press), January 1995, and (2) ITERATIVE COMPUTER ALGORITHMS with APPLICATIONS in ENGINEERING (Solving Combinatorial Optimization Problems): published by IEEE Computer Society Press, California, USA, 1999. He was the Head of Computer Engineering Department, KFUPM from January 2001–December 2004, Director of Information Technology and CIO of KFUPM between 2005 and 2011, and now is the Director of the Center for Communications and IT Research at the Research Institute of KFUPM.