Applied Soft Computing 14 (2014) 653–662

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A memory efficient stochastic evolution based algorithm for the multi-objective shortest path problem

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

[a] Faculty of Engineering, Gunma University, 29-1 Honcho, Ota, Gunma 373-0057, Japan
[b] Center for Communication and IT Resarch, Research Institute, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia
[c] Faculty of Engineering, Minia University, Minia, Egypt

ARTICLE INFO

Article history:
Received 23 December 2011
Received in revised form 12 August 2013
Accepted 17 September 2013
Available online 23 October 2013

Keywords:
Multi-objective shortest path
Multi-objective optimization
Stochastic evolution (StocE)

ABSTRACT

Multi-objective shortest path (MOSP) problem aims to find the shortest path between a pair of source and a destination nodes in a network. This paper presents a stochastic evolution (StocE) algorithm for solving the MOSP problem. The proposed algorithm is a single-solution-based evolutionary algorithm (EA) with an archive for storing several non-dominant solutions. The solution quality of the proposed algorithm is comparable to the established population-based EAs. In StocE, the solution replaces its bad characteristics as the generations evolve. In the proposed algorithm, different sub-paths are the characteristics of the solution. Using the proposed perturb operation, it eliminates the bad sub-paths from generation to generation. The experiments were conducted on huge real road networks. The proposed algorithm is comparable to well-known single-solution and population-based EAs. The single-solution-based EAs are memory efficient, whereas, the population-based EAs are known for their good solution quality. The performance measures were the solution quality, speed and memory consumption, assessed by the hypervolume (HV) metric, total number of evaluations and memory requirements in megabytes. The HV metric of the proposed algorithm is superior to that of the existing single-solution and population-based EAs. The memory requirements of the proposed algorithm is at least half than the EAs delivering similar solution quality. The proposed algorithms also executes more rapidly than the existing single-solution-based algorithms. The experimental results show that the proposed algorithm is suitable for solving MOSP problems in embedded systems.

1. Introduction

The multi-objective shortest path (MOSP) problem aims to find the shortest path between a pair of source and a destination node in a network. Two or more objectives should exist in a MOSP problem; for example, finding paths that minimize both travel time and total distance. MOSP with two or more objectives is an NP hard problem [1,2] that is relevant to many applications such as navigation systems of vehicles (including electric [3–5]) and unmanned aerial vehicles), robot path planning, and path selection in computer networks. Many applications execute on embedded systems with limited memory and computational speed. For instance, the navigation systems of intelligent vehicles operate in less powerful embedded systems to reduce power consumption. Intelligent vehicles include electric and hybrid vehicles, and vehicles based on internal combustion engines. Various types of algorithms are available for solving MOSP problems. To be suitable for embedded systems, an algorithm must find high quality solutions to the MOSP problem while consuming small memory space. In numerous applications, the parameter values in a graph or network are dynamic, the MOSP algorithm must also consider the changes in the underlying network. Dynamic changes occur, for example, in the traffic on a road network.

MOSP algorithms are of three different types: polynomial time exact algorithms (PTEAs), polynomial time approximation algorithms (PTAAs), and evolutionary algorithms (EAs). Although PTEAs find exact solutions to an MOSP problem [6], they consume much memory and computational time. Therefore, they are inappropriate for large networks. PTAAs are efficient in solving MOSP problems but are limited by two drawbacks: (i) they are sensitive to dynamic changes in the network and (ii) they are specific to one or few problems. EAs, which mimic the biological process of evolution to find optimal solutions, can also efficiently solve MOSP problems. In many EAs, the calculations in any generation are independent from those in previous generations. Therefore, EAs are sufficiently

* Corresponding author. Tel.: +81 276 50 2532.
E-mail addresses: umair@emb.cs.gunma-u.ac.jp, umair.farooq.siddiqi@gmail.com, ufarooq.geo@yahoo.com (U.F. Siddiqi), siraisi@emb.cs.gunma-u.ac.jp (Y. Shiraishi), mona@emb.cs.gunma-u.ac.jp (M. Dahb), sadiq@kfupm.edu.sa (S.M. Sait).
[1] Tel.: +81 276 50 2532.
[2] Tel.: +966 3 8601900; fax: +966 3 8603955.

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.09.008

robust to accommodate dynamic changes in the underlying network. Because EAs are defined in terms of evolutionary operators, they can be described in a generic format. A single EA can be applied to different types of optimization problems. For instance, a popular EA, the non-dominated sorting genetic algorithm-II (NSGA-II) [7] has been adopted in multi-objective electromagnetic optimization [8], service restoration in distribution systems [9], and optimal application mapping on NoC infrastructure [10]. The problem of finding shortest paths in huge networks has been successfully solved by genetic algorithm (GA) [11,12], which is faster than the conventional Dijskstra's algorithm.

EAs can be divided into two classes based on their memory sizes. The first class computes a single solution and is memory efficient. Examples of this class of EA are simulated evolution (SimE) and stochastic evolution (StocE) [13,14]. Single-solution-based EAs are suitable for embedded systems with limited memory and computational power. The second class of EAs generate a population of solutions. Although memory intensive, they can produce more accurate results than single-solution-based EAs. GA [13] is an example of population-based EA.

StocE is a general purpose iterative stochastic algorithm [13,14]. It resembles a biological evolutionary process in which the solution eliminates its bad characteristics as the generations evolve. Recently, the authors demonstrated that StocE produces higher quality results than many other single-solution-based EAs [15]. This paper proposes an enhanced StocE algorithm for solving the MOSP problem. The performance of the proposed algorithm is comparable to that of existing single-solution-based and population-based EAs. In the proposed algorithm, the characteristics are the sub-paths, where bad-sub-paths are replaced with more favorable ones from generation to generation. The proposed algorithm also maintains a set of pareto optimal solutions found during the optimization.

The performance of the proposed algorithm is comparable to well-known single-solution and population-based algorithms. The single-solution-based algorithms include straightforward StocE (std-StocE), multi-objective simulated annealing (MOSA) [16], and (1+1)-pareto archived evolutionary strategy ((1+1)-PAES) [17]. The population-based algorithms are NSGA-II [7], and micro-genetic algorithm (micro-GA) [18]. (1+1)-PAES, NSGA-II, and micro-GA are recognized for their strong performance in different applications [8–10,19,20]. The comparison with std-StocE highlights the advantage of the proposed StocE design over std-StocE. In each experiment, all algorithms were executed with an equal duration of 30 s. The quality of solutions from each algorithm was measured by the hypervolume (HV) metric [21]. The HV metric of the proposed algorithm is superior to that of existing single-solution and population-based EAs. The proposed algorithm requires more memory than the single-solution-based algorithms but less than the population-based EAs. The experimental results show that the proposed algorithm is suitable for solving MOSP problems in embedded systems.

This paper is organized as follows: Section 2 presents the relevant previous work and Section 3 describes the MOSP problem. The proposed algorithm is introduced in Section 4. Section 5 presents the experimental results and discussion, and conclusions are presented in Section 6.

2. Related work

This section discusses existing algorithms for solving MOSP problems. These algorithms are classified into PTAAs and EAs.

2.1. PTAAs

Mandow et al. [22] extended the A* search algorithm to find MOSP solutions. The new algorithm, named MOA*, is a heuristic search algorithm that finds non-dominated solutions. The search process in MOA* is guided by heuristic functions. When the guiding heuristic fails a certain bounding test, MOA* becomes unreliable and produces no useful solution. However, used with an appropriate set of heuristics, it is a reliable algorithm. Tsaggouris and Zaroliagis [23] proposed an improved fully polynomial time approximation scheme (FPTAS) algorithm for solving MOSPs, which resembles the multi-objective Bellman–Ford algorithm. This algorithm offers the best time complexity among the PTAAs..

Horoba [24] analyzed a simple evolutionary algorithm comprising a fitness function and a mutation operation, and found that it met the requirements of a fully polynomial time randomized approximation scheme. The runtime of this algorithm was comparable to that of Tsaggouris and Zaroliagis's algorithm. However, to determine the optimal path, conventional FPTAS requires precomputation of some values such as node labels containing the cost of links. If the cost of links changes to reflect dynamic changes in the network, some or many of the node labels become invalid. Thus, conventional FPTAS does not robustly accommodate dynamic changes and the shortest path calculation must be restarted several times when dynamic changes occur in the network.

2.2. Single-solution-based EAs

Smith et al. [16] proposed a MOSA algorithm. They introduced a transformation function that converts a multi-objective solution into an energy value. By comparing the energy value between two solutions, a solution can be assessed as better, equal, or worse than the alternative solution. To determine the energy of any solution, the algorithm counts the number of dominating solutions in the archive of the nondominated solution set (NDS). Solutions of higher energy are considered inferior. The maximum capacity of NDS is fixed.

Knowles and Corne [17] proposed a multi-objective local search algorithm, named (1+1)-PAES algorithm. The PAES has three components: (i) a candidate solution generator, (ii) a candidate solution acceptor, and (iii) an NDS archive. The candidate solution generator implements a simple mutation operation that creates a new solution, called the mutant solution, in each iteration by random mutation. The candidate solution acceptance function operates as follows. If the mutant solution dominates the existing solution then the current solution becomes the mutant solution. Otherwise, the following rules are applied: If the mutant solution occupies a less crowded region of the NDS than the existing solution, the current solution becomes the mutant solution. As in the MOSA algorithm, the NDS has a fixed maximum capacity. When the NDS becomes full, a solution is removed to make space for a new solution (c). First, the grid position of c in NDS is determined. If c occupies a less crowded region than any existing solution in NDS, the solution occupying the most crowded region is removed and c is inserted. The experimental results show that despite the simplicity of this algorithm, it has proven very effective for solving multi-objective optimization problems.

2.3. Population-based EAs

Coello and Pulido [18] proposed micro-GA for high quality multi-objective optimization. This algorithm stores solutions in two types of memories: the population memory and the NDS archive. The population memory is further divided into replaceable and nonreplaceable portions. The replaceable portion contains the dynamic solutions, while the nonreplaceable portion contains solutions that cannot be changed during the optimization. The micro-GA also contains a micro-GA cycle with a separate population, whose members are selected from the replaceable and nonreplaceable portions of the population memory. The micro-GA

cycle performs conventional GA operations, i.e., crossover and mutation, and executes over a smaller number of iterations. At the end of the micro-GA cycle, two new nondominated solutions are copied from its population to both the replaceable portion of the population memory and to the NDS archive.

Deb et al. [7] proposed NSGA-II for performing multi-objective optimization. While iterating, this algorithm seeks a different solution set by preserving two types of solutions: nondominated solutions and the most distinct solutions in the population. An iteration cycle consists of the following steps: (i) the population, including child chromosomes, is sorted by the nondominant count of their solutions. The chromosomes are assigned pareto front values such that the nondominated solutions are selected in the first front. The second front contains the nondominated solutions when the solutions of the first front have been removed from the population; (ii) the crowding distances of the chromosomes are based on the distances between the objective function values of the chromosomes. The crowding distance helps to establish uniform spreading of the pareto optimal front; (iii) the children (equal to or half the population size) are created by crossover and mutation operations and added to the population; (iv) the chromosomes for the population of the next iteration are selected from the combined population (i.e., the population and child chromosomes). Experimental results proved that this algorithm is very successful at finding different pareto-optimal solution sets for multi-objective optimization problems. Bora et al. [25] added greedy reinforcement learning (RL) to NSGA-II, which realizes parameter self-tuning. Their new algorithm, NSGA-RL, tunes four NSGA-II parameters-the probabilities and distribution indices of crossover and mutation operations-based on the results of previous generations. Although NSGA-RL is slower than NSGA-II, its results approach the NSGA-II results with the optimal parameter values.

3. Problem description

Given 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. The graph contains N_v vertices and N_e edges. An edge $e_i \in E$ is represented as $e_i=(n_x, n_y)$, where n_x and n_y are the start and end nodes of edge e_i, respectively, e_i is associated with up to K weights, represented as $\{e_i.w_1, e_i.w_2, e_i.w_3, \ldots, e_i.w_K\}$.

In the MOSP problem, a user should select a pair of source (s) and destination (d) nodes in the network ($s, d \in V$). A path between nodes s and d is denoted as: $S=\{e_1, e_2, \ldots, e_m\}$ such that $S \subseteq E$, $e_1=(s, n_x)$, $e_m=(n_y, d)$, $n_x, n_y \in V$, and $m \leq N_e$. If e_m and e_{m+1} are any two consecutive edges in P, then the start node of e_{m+1} should match the end node of e_m. The solution space U contains all possible paths between nodes s and d. U can become enormous in size in large networks. If d cannot be reached from s, U is an empty set.

In the MOSP problem, the objective function value of any path $S \in U$ is obtained by summing the weights of the edges in the path. The MOSP problem can contain up to K objective functions, which are represented as $f_k(S) = \sum_{e_x \in S} e_x.w_k$, for $k=1$ to K. The optimization goal can be formulated as follows:

$$Minimize(f_1(S), f_2(S), \ldots, f_K(S)) \quad (1)$$

A path $S_x \in U$ is said to strictly dominate (or simply dominate) another path $S_y \in U$ (represented as $S_x \prec S_y$) if the following two conditions are satisfied.

$$\forall i \in \{1, \ldots, K\} : f_i(S_x) \leq f_i(S_y) \quad (2)$$

$$\exists i \in \{1, \ldots, K\} : f_i(S_x) < f_i(S_y) \quad (2)$$

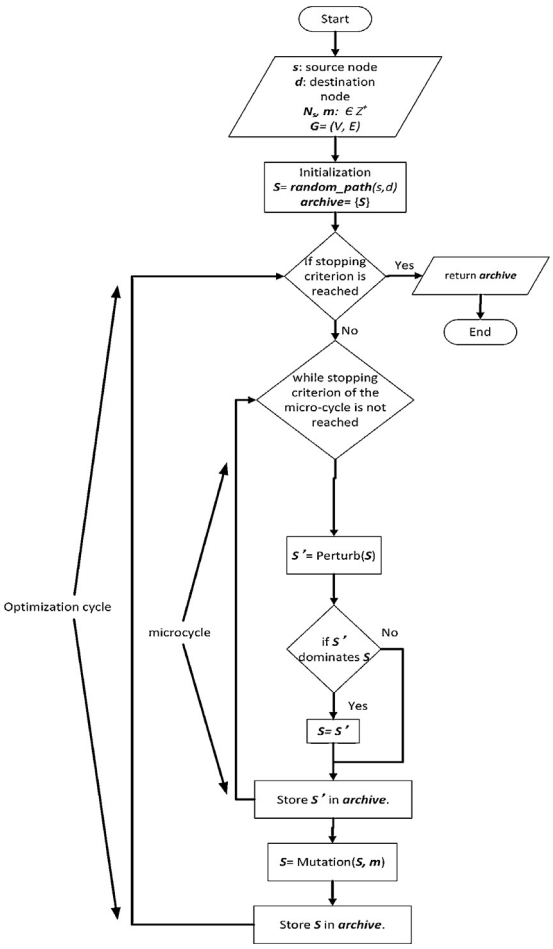

Fig. 1. Flowchart of the proposed algorithm.

The path S_x should be equal to or better than S_y in all K objective functions. In addition, S_x should be better than S_y in at-least one objective function.

A path $S_i \in S$ is said to be pareto-optimal w.r.t. U if and only if no other path in U dominates S_i. The set of all pareto-optimal solutions in U is called the pareto-optimal set. The corresponding set of objective function values is called the pareto-front, denoted by P^*. The set of non-dominated solutions found by any algorithm is denoted by Q.

The quality of solutions in Q of any algorithm can be determined from the HV metric [21,26,27], which calculates the area occupied by set Q. The HV measures both the quality and diversity of solutions. Higher quality pareto-optimal sets occupy a greater HV. The HV metric reliably determines the quality of solutions output by any algorithm.

4. Proposed algorithm

Our proposed algorithm generates a single solution and has the following salient features:

1 The characteristics of the Perturb operation are several random sub-paths in the current solution. This operation ranks the sub-paths w.r.t. their suitability to remain in the next generation. The least suitable sub-path is replaced by another randomly generated sub-path.
2 Mutation operation imposes a small probability that allows escape from the local minima.

Input: nodes: s & d
Output: y: A path from s to d nodes.
1: $Q = \emptyset$, $done = 0$, $y = \emptyset$
2: **for** (each node $n_i \in V$) **do**
3: $n_i.sel = false$, $n_i.\pi = 0$
4: **end for**
5: $Q = Q \cup \{s\}$
6: **while** (!$done$) **do**
7: η= A randomly selected element from Q, $Q = Q - \{\eta\}$
8: **for** (each node $n_x \in Adj(\eta)$) **do**
9: **if** ($n_x == d$) **then**
10: $n_x.\pi = \eta$, $done = 1$
11: **else if** (!$n_x.sel$ and $Q \cap \{n_x\}$) **then**
12: $n_x.\pi = \eta$, $Q = Q \cup \{n_x\}$, $n_x.sel = true$
13: **end if**
14: **end for**
15: **end while**
16: $n = d$, $y = y \cup \{n\}$
17: **while** ($n! = src$) **do**
18: $n = n.\pi$
19: $y = y \cup \{n\}$
20: **end while**
21: $Reverse\_Order(y)$
22: **return** y

Fig. 2. Method for finding a random path, $y = form\_path(s, d)$.

3 The non-dominated solutions found during the search operation are retained in an archive of solutions.

Our proposed StocE algorithm is shown in Fig. 1. The inputs to the algorithm are (i) source (s) and destination (d) nodes, (ii) maximum number of characteristics or sub-paths equal to N_s, (iii) a positive integer m for the mutation operation, and (iv) the network (G) in which the s and d nodes are selected. The current solution S is a complete path between nodes s and d. The first step is the initialization of the current solution S. *archive* is the stored set of all non-dominated solutions. Initially, *archive* contains only the current solution S. The next step is execution of an optimization cycle until the stopping criterion is reached. The stopping criterion can be the maximum number of iterations or the maximum execution time. The optimization cycle executes the microcycle, mutation operation, and an operation that stores the current solution S into *archive*, if S meets the storage requirements. Any new solution, i.e., S or S' can be added to *archive* unless it is not dominated by any existing solution in *archive*. The number of iterations in the microcycle should be less than the total number of iterations in the optimization cycle. The microcycle searches for better solutions through the Perturb operation on the current solution. As shown in Fig. 1, the Perturb operation generates the solution S', which replaces the existing current solution S if it dominates S. The solution S' is also stored in the *archive* if meets the storage requirements. The mutation operation in the optimization cycle allows the algorithm to escape from local minima. In each optimization cycle, the mutation operation generates a different value of the current solution for the microcycle.

The different steps in the algorithm are detailed in the following sections.

4.1. Initialization

In this step, the current solution S is initialized with a random solution. The function $form\_path(s, d)$ (Fig. 2) generates a random path between nodes s and d to be stored in S. In line 1, the set Q, a path y, and a variable *done* are initialized. The *for* loop between lines 2 and 4 initializes two attributes, *sel* and π, which are associated with all nodes for finding the random paths. The attribute *sel* of each node is initially set to *false*, but becomes *true* value when the associated node is inserted into Q. Once *sel* has been set to *true*, it remains *true* until the function terminates. The attribute π of each node stores the preceding node in the path from source to destination nodes. In line 5, source node s is added to Q. The *while* loop between lines 6 and 15 executes until the destination node is reached. A node η is randomly selected from Q. The *for* loop between lines 8 and 14 operates on the nodes adjacent to η, denoted as n_x. If n_x is the destination node, the attribute π of n_x is updated to η and the variable *done* is set to *true*, which causes the outer most *while* loop to terminate. If n_x is neither the destination node nor pre-existing in Q, then n_x is inserted into Q, and its attributes π and *sel* are set to η and *true*, respectively. The *while* loop between lines 17 and 20 forms a complete path in reverse order. Line 21 calls a function *Reverse_Order*, which reverses the order of nodes in y such that the last node s and the first node d become the first and last nodes, respectively.

4.2. Perturb operation

In StocE, each characteristic of the solution (in this case, each sub-path) should prove its suitability to remain in the next generation. This feature is implemented by the two-part Perturb operation. First, Perturb finds a sub-path in S that is minimally suited to remain in the next generation. The suitability of any sub-path p is determined by three quantities: (i) its objective function values ($f_1(p), f_2(p), \ldots, f_K(p)$) and (ii) the number of elements in the sub-path. Sub-paths with higher valued objective functions and fewer elements are considered less suitable. Second, the Perturb operation finds an alternative sub-path to replace the selected sub-path from S. This alternative sub-path replaces the selected sub-path if it dominates the selected sub-path.

Input: S: current solution, $N_s \in Z^+$
Output: $y = \{y_1, y_2\}$, starting (y_1) and ending (y_2) indices of a sub-path in S

1: st= null, en= null, NN= null, $rank$= null, n = Number of edges in S
2: **for** k= 1 to K **do**
3: $F_k = null$
4: **end for**
5: **for** i = 0 to $N_s - 1$ **do**
6: $i_1 = 0$, $i_2 = 0$
7: **while** $i_1 \geq i_2$ **do**
8: i_1= random integer between 0 and $n - 1$
9: i_2= random integer between $n - 1$
10: **end while**
11: **for** k=1 to K **do**
12: $F_k[i] = f_k(S[i_1....i_2])$
13: **end for**
14: $st[i] = i_1$, $en[i] = i_2$, $NN[i] = i_2 - i_1 + 1$
15: **end for**
16: **for** i=0 to N_s-1 **do**
17: $r_1 = 0$, $r_2 = 0$, ..., $r_{K+1} = 0$
18: **for** j=0 to N_s-1 **do**
19: **for** k=1 to K **do**
20: **if** $i! = j$ and $F_k[i] > F_k[j]$ **then**
21: r_k++
22: **end if**
23: **end for**
24: **if** $i! = j$ and $NN[i] < NN[j]$ **then**
25: $r_{K+1}++$
26: **end if**
27: **end for**
28: $rank[i] = \sum_{k=1}^{K} r_k + K \times r_{K+1}$
29: **end for**
30: I: index of the maximum element in $rank$
31: $y_1 = st[I]$, $y_2 = en[I]$, $y = \{y_1, y_2\}$
32: **return** y

Fig. 3. Function for selecting a sub-path, $y = select\_subpath(S, N_s)$.

Input: S: current solution, $N_s \in Z^+$, $G = (V, E)$
Output: S': solution after the Perturb operation

1: $I = \{i_1, i_2\} = select\_subpath(S, N_s)$
2: $e_1 = (n_a, n_b) = S[i_1]$, $e_2 = (n_c, n_d) = S[i_2]$
3: $t = form\_path(n_a, n_d)$
4: $S' = concatenate\{S[0...i_1 - 1], t, S[i_2 + 1....]\}$
5: **return** S'

Fig. 4. Perturb operation, $S' = Perturb(S, Ns)$.

Figs. 3 and 4, respectively, show the first and second parts of the Perturb operation. The first part inputs the current solution S and a positive integer N_s, which signifies the number of sub-paths to be considered in the allocation operation. It outputs the indices of the first (y_1) and last (y_2) edges of a sub-path in S. Line 1 initializes the arrays *st*, *en*, *NN*, and *rank*, each of which contains N_s elements. The variable n is set equal to the number of elements in S. The arrays F_k (where k= 1 to K), also containing N_s elements, are initialized in lines 2–4. The first *for* loop, between lines 5 and 15, fills the arrays F_k, *st*, *en*, and *NN*. This loop first selects a random sub-path in S, then finds the values of its objective functions (f_k; k= 1 to K) and stores them in F_k. The indices of the first and last edges of the selected sub-path are stored in *st* and *en*. The number of edges in the selected sub-path is stored in *NN*.

The *for* loop between lines 16 and 29 performs the following operations on each sub-path: (i) determines the values of the variables $\{r_1, r_2, \ldots, r_{K+1}\}$ and (ii) uses these variables to calculate the rank of the sub-path. Higher ranked sub-paths are regarded as less suitable for retention in the next generation. As shown in the code, the variables r_1 to r_K are assigned by comparing the objective function values of the currently selected sub-path with those of the remaining sub-paths. The variable r_{K+1} is assigned in the same manner but is based on the number of nodes rather than the objective functions. Line 28 determines the rank of the currently selected sub-path. In the rank formula, the first K terms belong to the objectives functions and the last term r_{K+1} specifies the number of edges. Therefore, to ensure approximately equal weighting between both types of terms, K is multiplied by r_{K+1}. In line 30, I contains the index of the highest ranked sub-path. In line 31, the sub-path corresponding to I is stored in y, where the components y_1 and y_2 contains, respectively, the start and end positions of the sub-path in S that is least suited to remain in the next generation.

Fig. 4 shows the second part of the Perturb operation. The inputs are the current solution S and N_s. This function perturbs the current solution S and returns the resulting new solution S'. Line 1 chooses

Input: S: current solution, $m \in Z^+$
Output: S: solution after the mutation operation
1: n= number of elements in S, $update$= 0
2: **for** i= 1 to m **do**
3: r_n= random integer between 0 and $n-1$
4: e_{r_n}= (n_a, n_b)= $S[r_n]$
5: $t = form\_path(n_a, d)$
6: $S' = concatenate\{S[0...r_n - 1], t\}$
7: **if** S' is not dominated by S **then**
8: $S = S'$, $update$= 1
9: **end if**
10: **end for**
11: **if** $update == 0$ **then**
12: $S = S'$
13: **end if**
14: **return** S

Fig. 5. Mutation operation, $S = mutation(S, m)$.

a sub-path in S by calling the function *select_subpath*, which implements the first part of the Perturb operation. In line 2, $e_1 = (n_a, n_b)$ and $e_2 = (n_c, n_d)$ store the first and last edge, respectively, of the selected sub-path. Line 3 constructs a new sub-path (t) between nodes n_a and n_d, while line 4 builds a new solution S' as follows: the new sub-path t is sandwiched between two portions of the path S. The first portion of S lies from the first element to the $(i_1 - 1)$th element, while the second portion lies from the $(i_2 + 1)$th to the final element. The last line of *Perturb* returns the new solution S'.

4.3. Storing non-dominated solutions

Our proposed algorithm maintains an archive of non-dominated solutions, designated as *archive*, which can store up to A solutions. The solutions in *archive* should be mutually non-dominated. When the archive becomes full, whether a new solution should displace an existing member in the archive is decided by the adaptive grid method proposed by Knowles et al. [17]. In this method, each solution in the archive is assigned a K-dimensional grid location (where K is the number of objectives). If the new solution resides in a less crowded grid location, it replaces an existing solution in a more crowded location. The range of each dimension of the grid is known from the archived solutions. The grid location of any solution is determined by recursively bisecting each dimension and finding the half in which the solution lies. The maximum number of subdivisions in each dimension is set by the user, for example, 15 or 25. Our proposed algorithm stores either the current solution S or the perturbed solution S' in *archive* via the procedure shown in Fig. 1.

4.4. Mutation

The mutation operation moves the current solution to another random location in the solution space. This operation aims to increase the diversity of solutions and allow escape from local minima. The loop in the mutation operator finds a new position that is not dominated by the current solution. If the loop fails to find a new non-dominated position, the current solution is updated to the path generated in the final iteration of the loop. The proposed mutation operation (Fig. 5) inputs the current solution S and a positive integer m, and outputs the solution obtained after the mutation operation. The value of m (usually small) specifies the number of iterations in the mutation loop. These iterations are used to find a new position that is not dominated by the current solution. Line 1 of mutation specifies n, the number of elements in S, and sets the variable *update* to 0. The *for* loop performs the following operations: (i) a random integer r_n is generated between 0 and $n-1$. In line 4, e_{r_n} defines the edge at position r_n in S, where n_a and n_b are the start and end nodes of e_{r_n}, respectively. Line 5 constructs a new sub-path (t) between nodes n_a and d. In lines 7 and 8, the current solution S is updated to the new solution S', unless S' is dominated by S. The *if* condition in lines 11 and 12 ensures that the mutation operation always replaces S by the new solution S' before termination of the operation.

Table 1
Characteristics of the graphs used in the simulation experiments.

Graph	Number of nodes (N_v)	Number of edges (N_e)
BAY	321,270	800,172
COL	435,666	1,057,066
NY	264,346	733,846

5. Simulation results

The tested algorithms, namely the proposed algorithm, NSGA-II, micro-GA, MOSA, (1+1)-PAES and std-StocE, were implemented in Visual Studio C# and run on an Intel i5 core, 2.27 GHz computer with 3 GB RAM. None of the programs used multi-processors or multi-threading. This section describes the experiments in detail and analyzes the results.

The experiments were performed on San Francisco Bay Area (BAY), Colorado (COL), and New York city (NY) road networks [28]. All of these road networks are huge as evident by the number of nodes and edges in each road network, as shown in Table 1. Each edge in the road network is assigned two weights: edge length and travelling time. Therefore, for any edge $e_i \in E$, the weights $e_i.w_1$ and $e_i.w_2$ are known. The algorithms solved a two-objective MOSP problem with the following objective function: $Obj(S) = Minimize(f_1(S), f_2(S))$, where $f_1(S)$ and $f_2(S)$ denote the total length and the total time required to travel the path S, respectively.

Each algorithm was set to terminate after 30 s. This execution time is intermediate for navigation problems in very large networks. The execution time of Dijsktra's algorithm in such networks is around 8 min [11]. The proposed algorithm was implemented under the following parameter settings: number of microcycle iterations = 6, *archive* size A = 10 solutions, number of sub-paths in the Perturb operation (N_s) = 6, and number of iterations in the mutation operation (m) = 3. The number of bisections in each dimension required to locate any solution in *archive* was set to 10. The solution size was kept variable.

NSGA-II was implemented with a population size of 50 solutions. The parents used in the crossover operation were selected by tournament selection based on crowding distance. During each iteration, the population members in the next iteration were selected from the population and children sets. Selection was based on the non-dominant count and diversity of solutions.

The micro-GA was implemented with a population memory of 40 solutions, of which 40% were non-replaceable and the remaining were replaceable. The population in the micro-GA cycle comprised 30 solutions, with nominal convergence reaching within 5 iterations. In the micro-GA cycle, the parents for the crossover operation were selected by tournament selection based on non-dominant count of the solutions. In the micro-GA cycle, a single non-dominated solution was retained from one generation to the next. Once nominal convergence of the micro-GA cycle was reached, one or two non-dominated solutions were copied to both external memory and replaceable portion of the population memory.

In both NSGA-II and micro-GA, the MOSP problem was solved using crossover and mutation operations as proposed by Ahn and Ramakrishna [12]. Crossover and mutation probabilities were set to 0.90 and 0.15, respectively.

Table 2
Symbols used for variant HV and HVR computations.

	Symbol	Description
1	*HV*(*Proposed*)	Average HV of the non-dominated set of the proposed algorithm
2	*HV*(*NSGA-II*)	Average HV of the non-dominated set of NSGA-II
3	*HV*(*Micro-GA*)	Average HV of the non-dominated set of Micro-GA
4	*HV*((1 + 1)-*PAES*)	Average HV of the non-dominated set of (1+1)-PAES
5	*HV*(*MOSA*)	Average HV of the non-dominated set of MOSA
6	*HV*(*std-StocE*)	Average HV of the non-dominated set of std-StocE
7	*HVR*(*NSGA-II*)	$\frac{HV(Proposed)}{HV(NSGA)}$
8	*HVR*(*Micro-GA*)	$\frac{HV(Proposed)}{HV(Micro\text{-}GA)}$
9	*HVR*((1 + 1)-*PAES*)	$\frac{HV(Proposed)}{HV((1+1)\text{-}PAES)}$
10	*HVR*(*MOSA*)	$\frac{HV(Proposed)}{HV(MOSA)}$
11	*HVR*(*std-StocE*)	$\frac{HV(Proposed)}{HV(std\text{-}StocE)}$

The mutation operation proposed Ahn and Ramakrishna [12] was also adopted in the (1+1)-PAES algorithm. In (1+1)-PAES, the archive contained 10 non-dominated solutions. The parameter l, specifying the number of bisections of each dimension in the adaptive grid, was set to 10.

The parameters in MOSA were set as follows: initial temperature (T_0) = 100, decreasing by 10% per iteration. The lower threshold of archived pareto optimal solutions was set to 4; i.e., if the archive of pareto optimal solutions contained less than four solutions, hypothetical solutions dominated by at least one solution were found by interpolation. The perturb operation in MOSA is similar to the mutation operation in GA [12].

A Std-StocE was also implemented. Although StocE has not been previously applied to MOSP problems, the straightforward implementation highlights the advantage of the proposed algorithm over std-StocE. The Perturb operation of std-StocE considers each edge in the solution and implements as follows: (i) a loop selects a different edge from the current solution in each iteration, (ii) the function *form_path* constructs a new sub-path from the start node of the selected edge to its destination node. The new sub-path is integrated with the original solution to form new solution. If the new solution dominates the original solution, it replaces the original solution with probability 0.80. The new solution unconditionally replaces the original solution with probability 0.2. If the original solution is updated before the *for* loop terminates, the algorithm exits both the *for* loop and the Perturb operation.

The test case was a pair of source (s) and destination (d) nodes randomly selected from the road network. Each experiment consisted of ten trials of the test case. The HV of the non-dominated sets in the test case were calculated using the tool proposed by Fonseca et al. [29], which adopts the slicing objectives algorithm [26] for accurate and rapid HV computation. The bounding point in the HV calculations was selected by Knowles method [30]. Knowles selected the bounding point as $b_j = \max_j + \delta(\max_j - \min_j)$, where b_j is the bounding value of the jth coordinate and $\max_j$ and $\min_j$ are the maximum and minimum values, respectively, of the jth coordinate in the non-dominated solutions. δ is assumed as 0.01. Here we report the average HV values of the ten trials.

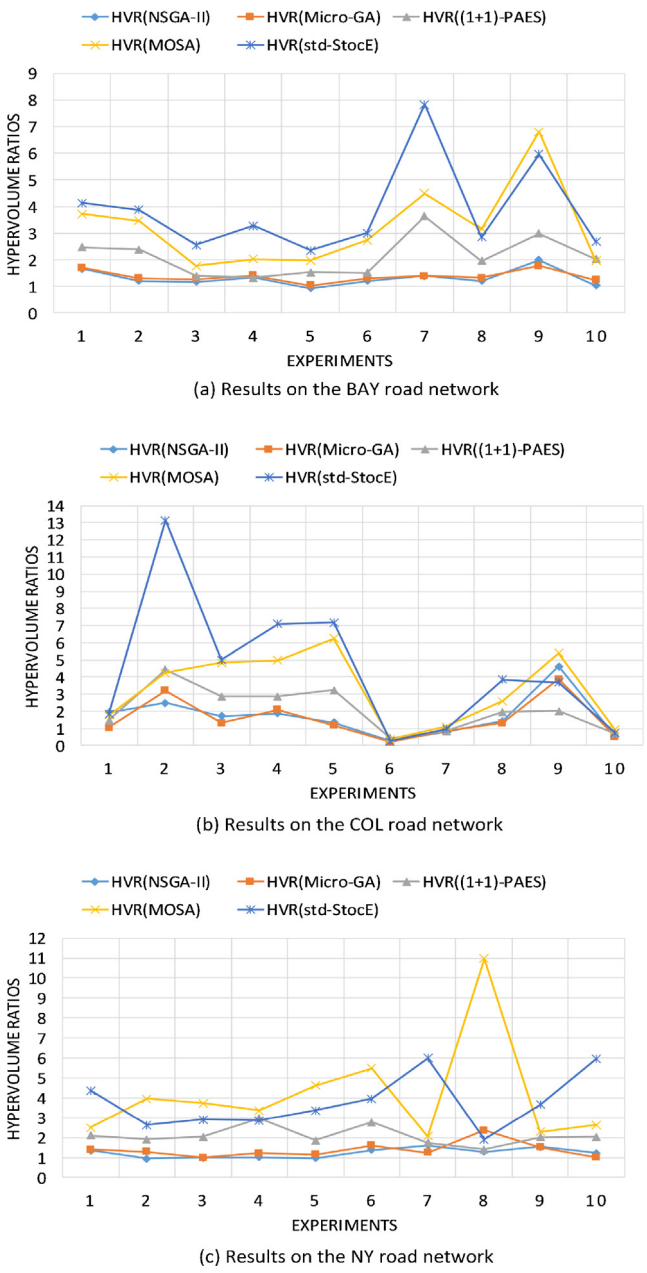

Fig. 6. Results of HVR calculations.

The HV values can vary considerably among different experiments. Therefore, the HV values of multiple experiments are difficult to display in a single graph. Instead, we collectively plot the ratios between the HV values of different algorithms (abbreviated as HVRs). Table 2 lists the various representations of HVs and HVRs employed in this study. The first six symbols are the average HV values of the ten trials in each experiment. The remaining five symbols are the HVRs between the proposed and the established algorithms. The HVR assesses whether the HV of the proposed algorithm is comparable to or surpasses that of the established algorithm. In this case, the HVR should be equal to or greater than one, respectively. Therefore, the HVR can efficiently compare the solution quality of the proposed algorithm with that of previously proposed algorithms in terms of their HV values.

Fig. 6 shows the results of the HVR calculations on different road networks. The x-axis shows the number of the experiments and the y-axis indicates the HVR values. For the NY road network (Fig. 6(a)), we find that (i) *HVR*(*NSGA-II*) ranges from 0.93 to 1.97 and

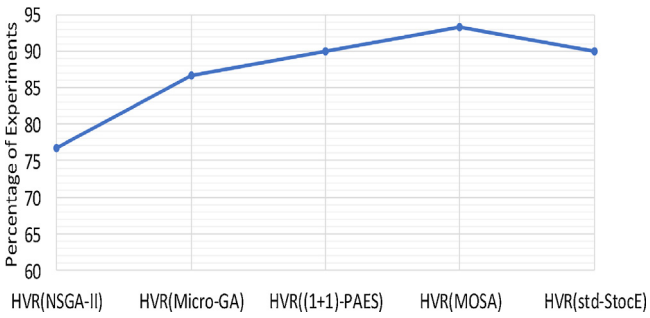

Fig. 7. Percentage of experiments in which $HVR \geq 1$.

equal or exceeds 1 in 80% of the experiments, while the other algorithms exceed 1 in all experiments, (ii) HVR(*micro-GA*) ranges from 1.02 to 1.77, (iii) HVR((1 + 1)-*PAES*) ranges from 1.33 to 2.99, (iv) HVR(*MOSA*) ranges from 1.77 to 6.80, and (v) HVR(*std-StocE*) ranges from 2.36 to 7.84. When the HVs of an algorithm x and the proposed algorithm are similar, the HVRs satisfy $1 < HVR(x) \leq (1 + \epsilon)$, where $0 < \epsilon < 1 \in R^+$, and $x \in \{$*NSGA-II*, *Micro-GA*, (1 + 1)-*PAES*, *std-StocE*$\}$. On the other hand, if ϵ is large, i.e., $\{\epsilon \geq 1, \epsilon \in R^+\}$ the HV of the proposed algorithm is at least doubly superior to algorithm x. From the graphs of Fig. 6, we observe that in most of the experiments, $HVR(x) \geq 1$, implying that the HV value of the proposed algorithm is comparable to that of the established algorithms.

The results of all HVR calculations are summarized in Fig. 7, which shows the percentage of experiments in which the $HVR(x)$ value exceeds 1. The HVRs of different algorithms are named along the x-axis, i.e., $HVR(x)$, where $x \in \{$*NSGA-II*, *Micro-GA*, (1 + 1)-*PAES*, *MOSA*, *std-StocE*$\}$. The y-axis indicates the percentage of experiments in which $HVR(x) > 1$; implying that the HV of the proposed algorithm is superior to that of algorithm x. We observe that the HV of the proposed algorithm is better than (i) NSGA-II in 77% of the experiments, (ii) micro-GA in 87% of the experiments, (iii) (1+1)-PAES in 90% of the experiments, (iv) MOSA in 93% of the experiments and (v) std-StocE in 90% of the experiments. Therefore, the solution quality of the proposed algorithm is higher than that of any existing algorithm tested ion this study.

The total number of evaluations of the objective function on a complete path performed by any algorithm echoes the number of solutions formed during the optimization. The objective function of the MOSP problem is not complex, and fast MOSP algorithms should (i) converge to an optimal solution using a reduced number of evaluations and (ii) execute more evaluations per unit time than the other algorithms. To determine the speed of the algorithms evaluated in this study, the total number of evaluations performed during their optimization within a fixed execution time of 30 s were measured. The results are listed in Table 3. In this table, the first and second columns specify the road network and the experiment number, respectively. The remaining columns provide the total number of evaluations performed by different algorithms. The evaluations were measured during the same experiments conducted for HV calculations. The total number of evaluations are averaged over the ten trials of each experiment, and expressed as multiples of z, where z is the total number of evaluations performed by the proposed algorithm. From these results, we infer that our proposed algorithm generally performs more evaluations and is faster than existing single-solution-based algorithms. We also note that the proposed algorithm reaches a good solution after moderate number of evaluations. In comparison to population-based algorithms, the proposed algorithm executes fewer evaluations in 30 s, yet converges rapidly to a high quality solution.

The memory requirements of the algorithms were also analyzed using Visual Studio's Profiler, which determines the amount of memory allocated to the different functions of each algorithm. The total memory consumption of the algorithms is the sum of the memory allocated to its various functions. The memory requirements were tested on the NY road network with a stopping criterion of 10 iterations. Each experiment contained just

Table 3
Average number of evaluations of the tested algorithms expressed as multiples of the proposed algorithm.

Road network	#	Total number of evaluations					
		Proposed$z$	NSGA-II($\times z$)	Micro-GA($\times z$)	(1+1)-PAES$\times z$	MOSA$\times z$	std-StocE$\times z$
BAY	1	186	9.38	18.33	0.74	0.73	0.15
	2	559	42.08	62.91	0.72	0.72	0.14
	3	496	29.87	49.36	0.71	0.73	0.12
	4	455	41	67.16	0.82	0.84	0.18
	5	433	34.26	56.89	0.75	0.79	0.11
	6	1216	58.05	99.24	0.90	0.87	0.10
	7	580	39.08	69.78	0.68	0.68	0.18
	8	702	49.74	86.93	0.62	0.62	0.12
	9	160	11.56	23.47	0.81	0.83	0.20
	10	247	16.06	27.11	0.66	0.68	0.13
COL	1	106	0.47	4.15	0.74	0.77	0.18
	2	237	5.50	10.97	0.77	0.78	0.13
	3	142	3.94	8.91	0.74	0.75	0.19
	4	372	24.32	35.66	0.78	0.77	0.16
	5	174	7.46	24.25	0.67	0.67	0.15
	6	137	0.68	9.34	0.74	0.75	0.15
	7	172	4.94	14.33	0.70	0.72	0.15
	8	414	33.30	63.02	0.69	0.67	0.10
	9	155	0.32	4.10	0.59	0.57	0.12
	10	186	9.32	23.09	0.70	0.76	0.15
NY	1	729	43.93	82.91	0.72	0.72	0.12
	2	144	12.76	22.33	0.75	0.77	0.19
	3	4303	62.60	150.96	1.15	1.14	0.32
	4	286	21.86	34.27	0.71	0.73	0.15
	5	313	24.20	35.62	0.72	0.75	0.14
	6	172	11.51	19.74	0.71	0.74	0.20
	7	438	29.04	57.07	0.67	0.67	0.09
	8	2906	61.93	131.09	1.13	1.10	0.23
	9	839	50.62	88.79	0.98	0.96	0.14
	10	400	38.08	60.99	0.88	0.88	0.16

Table 4
Memory consumption of the tested algorithms, expressed as multiples of the proposed algorithm (unit = megabyte (MB).)

#	Memory consumption					
	Proposed MB (y)	NSGA-II $\times y$	Micro-GA $\times y$	(1+1)-PAES $\times y$	MOSA $\times y$	std-StocE $\times y$
1	339.63	2.75	6.12	0.22	0.22	0.70
2	70.03	2.71	4.02	0.20	0.29	0.81
3	0.76	8.30	14.49	0.24	0.26	1.02
4	50.51	3.00	3.49	0.33	0.19	1.78
5	301.53	2.06	4.43	0.25	0.16	1.18
6	80.22	2.72	4.01	0.16	0.24	0.87
7	44.85	2.59	3.76	0.27	0.19	1.79
8	139.72	2.68	4.54	0.23	0.14	0.62
9	360.88	3.09	4.96	0.19	0.16	0.87
10	273.47	3.07	4.85	0.21	0.21	0.71

a single trial. In all of the tested algorithms, the populations or archives were of fixed size. Therefore, their memory requirements did not change significantly throughout the execution time. The results are presented in Table 4. The first column specifies the number of the experiment and the remaining columns show the memory consumption of the algorithms as multiples of the memory consumption of the proposed algorithm, denoted as y. The unit of memory consumption is megabytes (MB). From this table, we find that relative to the proposed algorithm, (i) NSGA-II requires 2–8 times more memory, (ii) micro-GA requires 3–14 times more memory, (iii) the (1+1)-PAES and MOSA algorithms require less memory (0.14–0.29 multiples), and (iv) std-StocE uses comparable memory (0.62–1.78 multiples). Therefore, the memory requirements of the proposed algorithm are at least half of the algorithms delivering similar solution quality, i.e., NSGA-II and micro-GA.

To summarize our experimental results, the proposed algorithm can produce higher quality solutions than many of the existing EAs. Because high quality solutions are found after comparatively few evaluations, it also executes more rapidly than other single-solution-based algorithms. The memory requirements of our proposed algorithm are at least half of those existing population-based algorithms.

6. Conclusion

We have proposed a StocE-based algorithm for solving the MOSP problem. The proposed algorithm outputs a single solution and its characteristics are different sub-paths in the solution space. In each iteration, a least suitable sub-path is replaced by a randomly generated sub-path. The performance of the proposed algorithm was compared with that of two population-based algorithms (NSGA-II and micro-GA) and three single-solution-based algorithms ((1+1)-PAES, MOSA, and std-StocE) on real road networks. The performance measures were the solution quality, speed, and memory consumption, assessed by the HV metric, total number of evaluations of the objective function within a specified time, and memory requirements in megabytes, respectively. More the evaluations performed within a specified time, more the solutions generated during the optimization. Within an appropriate execution time (in this case, 30 s) the solution quality and execution speed of the proposed algorithm surpassed that of many existing EAs. The execution speed is reduced because the proposed algorithm converges to a good solution after few evaluations. The memory requirements are reduced to at least half of those established population-based algorithms. Based on the experimental results, we conclude that the proposed algorithm is applicable to MOSP problems in embedded systems, which are generally implemented in simple processors with limited memory.

References

[1] Z. Tarapata, Selected multicriteria shortest path problems: an analysis of complexity, models and adoption of standard algorithms, Int. J. Appl. Math. Comput. Sci. 17 (2) (2007) 269–287.
[2] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., New York, 1997.
[3] U.F. Siddiqi, Y. Shiraishi, S.M. Sait, Multi-constrained route optimization for electric vehicles using SimE, in: International Conference on Soft Computing and Pattern Recognition (SoCPaR 2011), 2011, pp. 376–383.
[4] U.F. Siddiqi, Y. Shiraishi, S.M. Sait, Multi-constrained route optimization for electric vehicles (EVs) using particle swarm optimization (PSO), in: 11th International Conference on Intelligent Systems Design and Applications (ISDA), 2011, pp. 391–396.
[5] U.F. Siddiqi, Y. Shiraishi, S.M. Sait, Multi-objective optimal path selection in the electric vehicles, in: International Symposium on Artificial Life and Robotics (AROB), vol. 17, 2012.
[6] E. Martins, J. Santos, The Labelling Algorithm for the Multiobjective Shortest Path Problem, Departamento de Matematica, Universidade de Coimbra, TR-99/005, Portugal, 1999.
[7] K. Deb, A. Pratap, S Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[8] L. dos Santos Coelho, P. Alotto, Multiobjective electromagnetic optimization based on a nondominated sorting genetic approach with a chaotic crossover operator, IEEE Trans. Magnet. 44 (6) (2008) 1078–1081.
[9] Y. Kumar, B. Das, J. Sharma, Service restortation in distribution system using non-dominated sorting genetic algorithm, Electr. Power Syst. Res. 76 (2006) 768–777.
[10] M.V.C. da Silva, N. Nedjah, L. de Macedo Mourelle, Optimal application mapping on NoC infrastructure using NSGA-II and MicroGA, in: International Conference on Intelligent Engineering Systems (INES), 2009, pp. 83–88.
[11] A. Gunichev, S. Bedathur, S. Seufert, G. Weikum, Fast and accurate estimation of shortest paths in large graphs, in: Proc. 19th ACM International Conference on Information and Knowledge Management, Toronto, Canada, 2010, pp. 499–508.
[12] C.W. Ahn, R.S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, IEEE Trans. Evol. Comput. 6 (6) (2002) 566–579.
[13] S.M. Sait, H. Youssef, Iterative Computer Algorithms with Applications in Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1999.
[14] Y.G. Saab, V.B. Rao, Stochastic evolution: a fast effective heuristic for some generic layout problems, in: 27th Design Automation Conference, 24–28 June, 1990, pp. 31–36.
[15] U.F. Siddiqi, Y. Shiraishi, M. Dahb, S.M. Sait, Finding multi-objective shortest paths using memory-efficient stochastic evolution based algorithm, in: 3rd International Conference on Networking and Computing (ICNC), Okinawa, Japan, 2012, pp. 182–187.
[16] K.I. Smith, R.M. Everson, J.E. Fieldsend, C. Murphy, R. Misra, Dominance-based multiobjective simulated annealing, IEEE Trans. Evol. Comput. 12 (3) (2008) 323–342.
[17] J.D. Knowles, D.W. Corne, Approximating the nondominated front using the pareto archived evolution strategy, Evol. Comput. 8 (2) (2000) 149–172.
[18] C.A. Coello Coello, G.T. Pulido, A Micro-genetic Algorithm for Multiobjective Optimization, Departamento de Matematica, Universidade de Coimbra, TR-99/005, Portugal, 1999.
[19] A.R.M. Rao, K. Lakshmi, Multi-objective scatter search algorithm for combinatorial optimization, in: 16th International Conference on Advanced Computing and Communication (ADCOM), 2008, pp. 303–308.
[20] J. Teo, P. Anthony, J.H. Ong, Neural network ensembles for video game AI using evolutionary multi-objective optimization, in: 11th International Conference on Hybrid Intelligent Systems (HIS), 5–8 December, Malaysia, 2011, pp. 510–605.
[21] J.M. Bader, Hypervolume-based Search for Multiobjective Optimization: Theory and Methods, Swiss Federal Inst. Technology (ETH), Zurich, Switzerland, 2009 (Ph.D. dissertation).

[22] L. Mandow, J.L. Perez de la Cruz, A new approach to multiobjective A* search, in: Proc. of IJCAI'05, 2005, pp. 218–223.
[23] G. Tsaggouris, C. Zaroliagis, Multiobjective optimization: improved FPTAS for shortest paths and non-linear objectives with applications, J. Theory Comput. Syst. 45 (1) (2009) 162–186.
[24] C. Horoba, Exploring the runtime of an evolutionary algorithm for the multi-objective shortest path problem, Evol. Comput. 18 (3) (2010) 357–381.
[25] T.C. Bora, L. Lebensztajn, L.D.S. Coelho, Non-dominated sorting genetic algorithm based on reinforcement learning to optimization of broad-band reflector antennas satellite, IEEE Trans. Magnet. 48 (2) (2012) 767–770.
[26] L. White, P. Hingston, L. Barone, S. Husband, A faster algorithm for calculating hypervolume, IEEE Trans. Evol. Comput. 10 (1) (2006) 29–38.
[27] P. Ngatchou, A. Zarei, M.A. El-Sharkawi, Pareto multiobjective optimization, in: Proc. 13th Intelligent Systems Application to Power System, 2005, pp. 84–91.
[28] 9th DIMACS Implementation Challenge – Shortest Paths. URL: http://www.dis.uniroma1.it/challenge9/download.shtml
[29] C.M. Fonseca, L. Paquete, M. Lpez-Ibez, An improved dimension – sweep algorithm for the hypervolume indicator, in: 2006 IEEE Congress on Evolutionary Computation (CEC'06), 2006, pp. 1157–1163.
[30] J. Knowles, ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problem, IEEE Trans. Evol. Comput. 10 (1) (2005) 50–66.