Artif Life Robotics
DOI 10.1007/s10015-012-0025-5

ORIGINAL ARTICLE

Multi-objective optimal path selection in electric vehicles

Umair Farooq Siddiqi · Yoichi Shiraishi · Sadiq M. Sait

Received: 6 March 2012 / Accepted: 19 June 2012

Abstract This work proposes a memory-efficient multi-objective optimization algorithm to perform optimal path selection (OPS) in electric vehicles (EVs). The proposed algorithm requires less computational time and executes efficiently on fast-processor-based embedded systems. It is a population-based simulated evolution algorithm that incorporates innovative functions for calculating the goodness of particles and performing the allocation operation. The goodness and allocation operations ensure the exploration of new paths and the preservation of Pareto-optimal solutions. We executed our algorithm on an Intel Celeron processor, which is also used in embedded systems and compared its performance with that of the non-dominated sorting genetic algorithm-II (NSGA-II). Our experiments used real road networks. The comparison shows that on an average, our algorithm found 5.5 % more Pareto-optimal solutions than NSGA-II. Therefore, our proposed algorithm is suitable for performing OPS in EVs.

Keywords Multi-objective shortest path (MOSP) · Simulated evolution (SimE) · Electric vehicles (EVs)

This work was presented in part at the 17th International Symposium on Artificial Life and Robotics, Oita, Japan, January 19–21, 2012.

U. F. Siddiqi (✉) · Y. Shiraishi
Department of Production Science and Technology, Gunma University, Ota, Gunma, Japan
e-mail: umair@emb.cs.gunma-u.ac.jp

S. M. Sait
Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

1 Introduction

Navigation systems of modern vehicles are equipped with optimal path selection (OPS) units, which are responsible for finding shortest paths between source and destination nodes in a road network. The increasing popularity of electric vehicles (EVs) [1] demands that OPS units should also meet the requirements of EVs. The criteria for selecting shortest paths in EVs are different from the criteria used in internal combustion engine (ICE) vehicles. The criteria generally used for shortest paths in ICE vehicles are traveling time and distance. In contrast, OPS units in EVs should include recharging time in their criteria for shortest paths in addition to traveling time and distance.

The two main obstacles that hamper the popularity of EVs are their traveling limits and recharging time. The traveling limit of an EV is the maximum time that it can travel without recharging. Rechargeable batteries with higher capacities and greater ranges have been proposed to increase the traveling limits of EVs. Rapidly rechargeable batteries that can be recharged in as short a time as 10 min are currently available [2, 3]. However, suitably placed recharging stations will be the primary means of enabling EVs to travel distances that significantly exceed their traveling limits. Several researchers have investigated the feasibility of placing an adequate number of recharging stations in cities and have proposed several ways to do this [3, 4]. The rapid progress in EVs and EV-related fields will soon bring about the infrastructure that is required for trouble-free use of EVs. This makes it even more important to develop efficient OPS units that meet the requirements of EVs.

A car's OPS unit is situated in its navigation system. Car navigation systems are generally connected with the global positioning system (GPS) or V2X communication systems [5] in order to obtain information related to road networks. An OPS unit uses the information in the navigation system to determine shortest paths. There can be more than one objective in the shortest path problem; hence, the problem of finding shortest paths with OPS units is a multi-objective shortest path (MOSP) problem. MOSP problems are NP-hard discrete optimization problems [6, 7]. The objectives in an MOSP problem can be contradictory, and therefore, a set of Pareto-optimal solutions is required. A Pareto-optimal set is a subset of the solutions that can be found by any algorithm; it includes all solutions that are not dominated by other solutions.

Evolutionary computational (EC) algorithms have been predominately used to solve multi-objective optimization problems. EC algorithms differ from each other in the manner they mimic the biological process of evolution and adapt their parameters during their searches. EC algorithms can operate on a single solution or on a population of solutions. The population-based algorithms are suitable for finding Pareto-optimal solutions to multi-objective optimization problems because they simultaneously optimize several solutions. Genetic algorithms (GAs) and their variants, along with particle swarm optimization (PSO), are among the most well-known population-based algorithms. Simulated evolution (SimE) and simulated annealing (SA) [8] are two popular single-solution algorithms. SimE is fast and has been shown to be efficient in solving path optimization problems [9]. In contrast, population-based EC algorithms produce good results but generally require a large amount of memory. This paper presents a population-based SimE algorithm for solving the MOSP problem in EVs.

The purpose of developing population-based SimE algorithms is to improve the quality of the solutions produced by single-solution SimEs. Our proposed algorithm is memory efficient and requires only an amount of memory that is directly proportional to the population size. Conventional algorithms which include Dijkstra's algorithm, A* [10], and Martin's algorithm [11] for MOSP problem work well for small size networks, however, in huge size networks, their computational times increase significantly. The OPS units of EVs require memory efficient and fast algorithms. Two important features that distinguish our proposed algorithm with the other algorithms are, it is memory efficient than the previous EC algorithms and it can compute the MOSPs in huge size networks in short time because unlike conventional algorithms it does not need to visit all nodes in the road network. We compare the performance of our proposed algorithm with that of the non-dominated sorting genetic algorithm-II (NSGA-II). NSGA-II is one of the most successful and widely used multi-objective evolutionary computation algorithms. It has been found to have a very promising performance in solving multi-objective optimization problems in many different applications [12, 13]. Therefore, NSGA-II can act as a benchmark algorithm [14]. Our comparison shows that given the same amount of execution time and memory usage, our proposed algorithm performs better than NSGA-II. Therefore, our algorithm is suitable for solving the OPS problem in EVs.

The rest of this paper is organized as follows: Sect. 2 discusses previous related work. Section 3 formally describes the MOSP problem for EVs. Section 4 contains a detailed description of our proposed algorithm. Section 5 shows the calculation of the memory requirements of our algorithm. Section 6 presents our simulation results and compares our proposed algorithm with NSGA-II, and finally, Sect. 7 concludes the paper.

2 Related work

This section briefly describes some existing algorithms for solving the multi-objective optimization problem, existing research on the suitable placement of recharging stations, and existing research on parallel SimE algorithms.

Elitist evolutionary multi-objective optimization (EMO) algorithms are the most recently introduced EMOs for finding Pareto-optimal solution sets. Elitist EMO algorithms preserve the good solutions during their iterations. Some examples of popular elitist EMOs are as follows. Deb et al. [15] proposed the elitist multi-objective genetic algorithm NSGA-II, which has low computational complexity. During its iterations, it preserves two types of solutions: non-dominated solutions and solutions that are the most distinct in the population. By doing so, the algorithm maintains both quality and diversity among its solutions. Experimental results have shown that this algorithm is very successful in finding diverse Pareto-optimal sets of solutions for multi-objective optimization problems. Bora et al. [16] added greedy reinforcement learning to NSGA-II for self-tuning its parameters. Their new algorithm, NSGA-RL, tunes four NSGA-II parameters—the probabilities of crossover and mutation operations and the distribution indexes in crossover and mutation operations—on the basis of the results of previous generations. NSGA-RL is slower than NSGA-II; however, its results are closer to the NSGA-II results that have the best possible parameter values. Li [17] proposed a multi-objective optimization algorithm based on PSO called "Non-dominated Sorting Particle Swarm Optimization" (NSPSO). In NSPSO, any one non-dominated particle is considered to be the global best position and is used for calculating particle velocities. Experimental results show that NSPSO is competitive with NSGA-II.

Kriz et al. [18] investigated the placement of recharging stations on road networks. They proposed that the traffic flow on a road network should be known to determine the optimal placement of recharging stations. Recharging stations have been optimally placed when the maximum number of unique paths in a network with non-zero traffic flow contains recharging stations. Kriz et al. determined the locations using integer linear programming (ILP) optimization in which the total number of recharging stations is also constrained.

To date, only a few studies have addressed the development of parallel SimE algorithms. Sait et al. [19] presented an interesting overview of three types of parallel strategies for SimE algorithms. The first type evaluates moves in parallel, with traversal of a solution through the search space remaining unaltered. The second type partitions a complete solution into smaller subsolutions that are optimized in parallel. At the end of each iteration, subsolutions are combined to form a complete solution and then re-partitioned for the next iteration. The third type of parallel strategy performs multiple searches in the search space. Our proposed algorithm is based on this third type, with the elements in the population independently exploring the search space.

3 Description of the problem

3.1 Description of the road network

Let us assume that the road network is represented by an undirected graph $G = (V, E, Q)$, where V is the set of nodes, E is the set of edges, and Q contains the EVs that are traveling on the road network. The nodes in the set $V = \left\{n_0, n_1, \ldots, n_{N_{\mathrm{v}-1}}\right\}$, where N_{v} is the total number of nodes, represent the intersections in the road network and the edges in the set $E = \left\{e_0, e_1, \ldots, e_{N_{\mathrm{e}-1}}\right\}$, where N_{e} is the total number of edges, represent the roads that join the intersections. The nodes also contain recharging stations at which EVs can be recharged. The members of the set $Q = \left\{q_0, q_1, \ldots q_{N_{\mathrm{EV}-1}}\right\}$, where N_{EV} is the total number of EVs in the road network, represent the EVs. The road network is illustrated in Fig. 1. The navigation systems of the EVs in the road network can communicate with the traffic control station through GPS or V2I communication.

The properties of the nodes, edges and EVs are shown in Table 1. The symbols n_i, e_j and q_k represent elements from the sets V, E and Q, respectively. The property $n_i \times R_T$ is the time that the recharging station at node n_i takes to fully recharge an EV. The properties of each edge e_j include $e_j \times s_t$, which is the starting node of the edge, $e_j \times e_n$, which is the ending node of the edge, $e_j \times l$, which represents the length of the edge, and $e_j \times S_T$, which is the average traveling time along the edge e_j. The property $q_k \times T_l$ is the traveling limit of the EV q_k, i.e., the maximum time that q_k can travel without recharging, and the property $q_k \times B_s$ is the battery charge of q_k at the source node, which can range between 0 and 100 %.

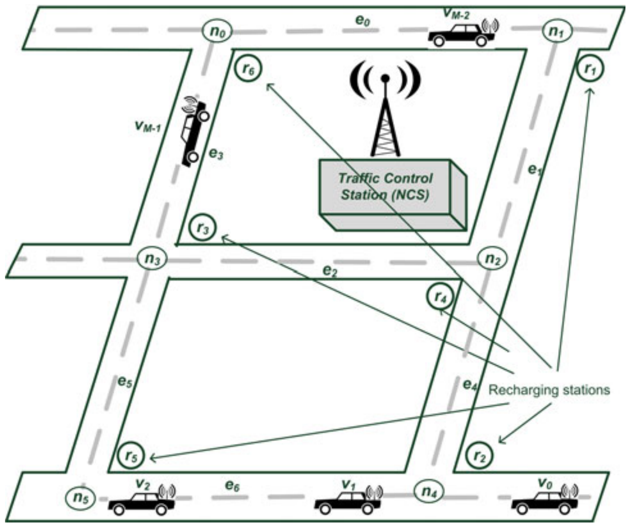

Fig. 1 Illustration of the road network for intelligent vehicles

Table 1 Properties of nodes, edges and electric vehicles

Symbol	Description	Range of values
Properties of any node $n_i \in V$		
$n_i \times R_T$	Recharging time for the EVs	$\{x \in \mathbf{R}^+\}$
Properties of any edge $e_j \in E$		
$e_j \times s_t$	Starting node of the edge	$\{x \in V\}$
$e_j \times e_n$	Ending node of the edge	$\{x \in V\}$
$e_j \times S_T$	Average traveling time on it	$\{x \in \mathbf{R}^+\}$
$e_j \times l$	Length of the edge	$\{x \in \mathbf{R}^+\}$
Properties of any electric vehicle $q_k \in Q$		
$q_k \times T_l$	Traveling limit of the EV	$\{x \in \mathbf{R}^+\}$
$q_k \times B_s$	Battery level at the source node	$\{x \in \mathbf{Z} \vert 0 < x \leq 100\}$

3.2 Description of the MOSP problem

We assume that individual EV will independently solve the MOSP problem through their OPS units. Figure 2 shows a block diagram of the proposed EV OPS units. The OPS unit receives information about the road network through GPS or V2X communication and is told the starting and ending nodes of the journey by the driver and it sends its output to the navigation system display unit.

The MOSP problem that the OPS unit of the EV q_k must solve is as follows. Assume that q_k desires to travel

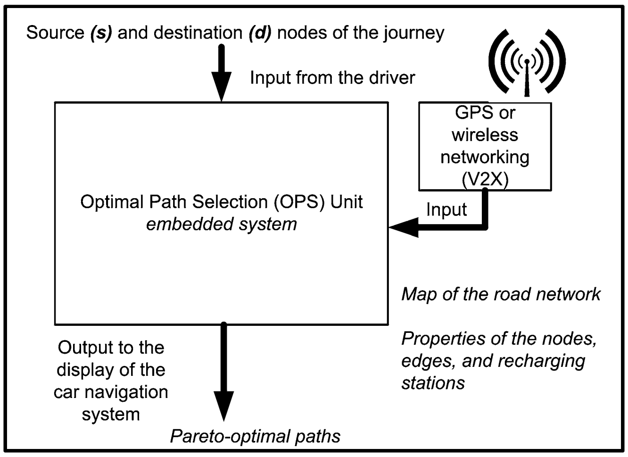
Source (s) and destination (d) nodes of the journey
Input from the driver
GPS or wireless networking (V2X)
Input
Optimal Path Selection (OPS) Unit embedded system
Map of the road network
Properties of the nodes, edges, and recharging stations
Output to the display of the car navigation system
Pareto-optimal paths

Fig. 2 Block diagram of the optimal path selection unit of the electric vehicles

between the source node s and destination node d, (where $s, d \in V$). The solution will be a set of Pareto-optimal solutions. Let $PT_K = \{P_0, P_1, \ldots P_{M-1}\}$, where M is the total number of Pareto-optimal solutions, be the Pareto-optimal set for the EV q_k. Each $P_j \in PT_k$ is a solution that consists of two parts represented as $P_j = \{P^A, P^B\}$, where $P^A \subseteq E$ stores a path between the nodes s and d and $P^B \subset V$ stores the nodes at which q_k should be recharged. The nodes in P^B are starting nodes of some of the edges in P^A. The value of any element $p_i \in P^A$ can be determined as follows:

$$p_i = \begin{cases} e_x \in E, \text{ s.t. } e_x \times s_t = s & \text{if } i = 0 \\ e_x \in E, \text{ s.t. } e_x \times s_t = p_{i-1} \times e_n & \text{if } i > 0 \\ \text{null} \quad \text{if } p_{i-1} \times e_n = d \text{ or } p_{i-1} = \text{null} \end{cases}$$

A solution is feasible if the length of each edge in the solution is equal to or less than the traveling limit of the EV q_k, i.e., $e_x \times l \le q_k \times T_l \, \forall e_x \in E$. Each solution P_j has an attribute *ParetoOptimal*, and this attribute is true only if P_j is a Pareto-optimal solution.

The proposed MOSP problem is a minimization problem that has three objective functions:

Minimize($f_1(P_j)$, $f_2(P_j)$, $f_3(P_j)$), where

$f_1(P_j)$ is the time consumed in recharging the EV on path P_j,
$f_2(P_j)$ is the total length of path P_j, and
$f_3(P_j)$ is the total traveling time on path P_j.

The values of the functions $f_k(P_j)$, $1 \le k \le 3$, are calculated as follows:

$$f_1(P_j) = \sum_{\forall n_x \in P^B} n_x \times R_T \tag{1}$$

$$f_2(P_j) = \sum_{\forall e_x \in P^A} e_x \times l \tag{2}$$

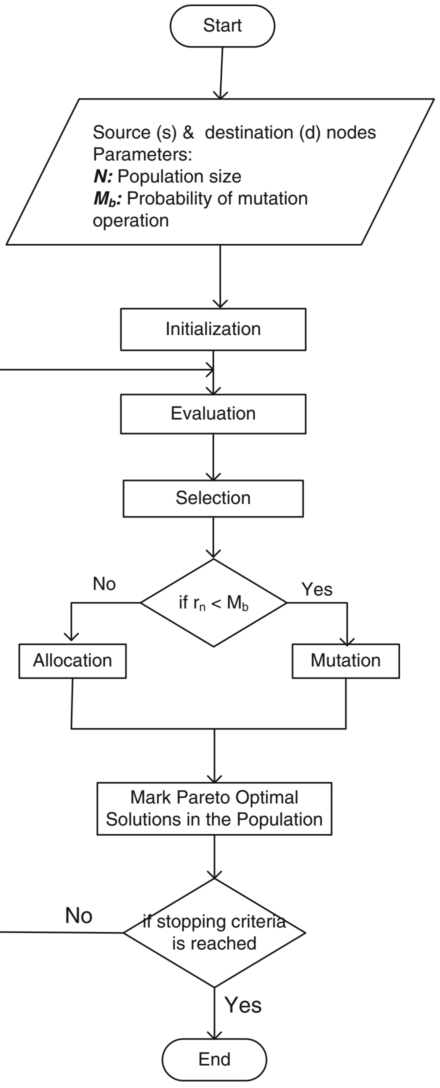
Start
Source (s) & destination (d) nodes
Parameters:
N: Population size
M_b: Probability of mutation operation
Initialization
Evaluation
Selection
if r_n < M_b
No
Yes
Allocation
Mutation
Mark Pareto Optimal Solutions in the Population
if stopping criteria is reached
No
Yes
End

Fig. 3 Proposed algorithm

$$f_3(P_j) = \sum_{\forall e_x \in P^A} e_x \times S_T. \tag{3}$$

4 Proposed algorithm for the MOSP problem

This section describes our proposed algorithm. There are four user inputs: s and d, the source and destination nodes for the EV's journey, the probability of mutation M_b, which should be a value in the range $\{x \in \mathbf{R} | 0 \le x \le 1\}$, and the population size N. The value of M_b should be kept small because high mutation probabilities lead to random searches. The steps that comprise the proposed algorithm are shown in Fig. 3. The stopping criterion can be the maximum number of optimization iterations or the maximum execution time for the iterations. The following subsections describe each step in detail. In addition to allocation, our algorithm also uses the mutation operation to explore new solutions. The allocation operation mutates a solution by altering the elements of the selection set [8]. On the other hand, mutation operation mutates a solution by altering randomly selected elements. The operation Mark Pareto-optimal solutions, is also introduced in order to preserve Pareto-optimal solutions from one generation to next generation. The preservation of Pareto-optimal solutions is known as elitism and many latest EC algorithms use it to maximize the number of Pareto-optimal solutions in their final solution set.

4.1 Initialization

In this first step, maximum N solutions are initialized to random paths between the source and destination nodes. The population is denoted by $\mathrm{POP} = \{P_0, P_1, \ldots, P_{N-1}\}$. Any $P_j \in \mathrm{POP}$ can be represented as $P_j = \{P^{\mathrm{A}}, P^{\mathrm{B}}\}$. P^{A} is a complete path from the source to the destination nodes. P^{B} contains the recharging stations at which the EV should be recharged in order to complete its journey on the path P^{A}. The function *form_path(s,d)*, shown in Fig. 4, is used to generate a random path between the nodes s and d. In line 1, a set Q, a path y and a variable *done* are initialized. The *for* loop between lines 2 and 4 performs the initialization of the two attributes (*sel* and π) that are associated with all nodes for the purpose of finding the random paths. The attribute *sel* of each node is initially set to *false* (or *0*), but it changes to *true* value when the associated node is inserted into Q. After changing to *true*, its value remains *true* until the end of the function. The attribute π for each node stores the node that is preceding it in the path from source to destination nodes. In line 5, source node (*src*) is added to Q. The *while* loop between the lines 6 and 21 executes until the destination node is reached. η is a node which is randomly selected from Q. The *for* loop between lines 8 and 14 selects the nodes that are adjacent to η. n_x represents a node which is adjacent to η. If n_x is the destination node then the attribute π of n_x is updated to η and the value of variable *done* is set to *true* which causes the outer-most *while* loop to terminate. If n_x is not equal to the destination node and in addition n_x is never inserted in Q then n_x is inserted into Q. When n_x is inserted into Q, the value of its *sel* is set to *true*. The *while* loop between lines 17 and 20 forms a complete path in the reverse order. Line 21 calls a function *Reverse_Order* to correct the order to nodes in y which is from the source to the destination node.

The function *selectStations(P^{A})*, shown in Fig. 5, selects recharging stations on the path P^{A}. In line 1, P^{B} is initialized to null, and *CB* is initialized to the distance that the EV can travel with the battery level it has at the source node. In line 2, L_p stores the total length of path P^{A}.

```
Input: nodes: $s$, & $d$, $G = (V, E)$
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: $Q = Q \cup \{src\}$
 6: while ($!done$) do
 7:     $\eta =$ A randomly selected element from Q
 8:     for (each node $n_x \in Adj(\eta)$) do
 9:         if ($n_x == dest$) then
10:             $n_x.\pi = \eta$, $done = 1$
11:         else if ($!n_x.sel$ and $Q \cap \{n_x\}$) then
12:             $n_x.\pi = \eta$, $Q = Q \cup \{n_x\}$, $n_x.sel = 1$
13:         end if
14:     end for
15: end while
16: $n = dest$, $y = y \cup n$
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. 4 Function to find a random path, $y =$ *form_path(s,d)*

The *while* loop between lines 3 and 14 executes until *CB* contains a value that is greater than or equal to the total length of the path P^{A} (L_p). Line 4 initializes variables. The *for* loop between lines 5 and 11 populates the sets P^{s}, d^{s} and a^{s}. The set P^{s} contains the ending nodes of edges in P^{A} such that the length of the sub-path from the source node to those nodes is between L_s and *CB*. The set d^{s} stores the distances of nodes in P^{s} from the source node. For each node in P^{s}, the set a^{s} stores the sum of the recharging time and the reciprocal of the distance of that node from the source node. Line 12 sets I_{sel} to contain the index of the element in a^{s} that has the minimum value and V_{sel} to contain the value of the element at position I_{sel} in d^{s}; moreover, it adds the node at position I_{sel} in P^{s} to the set P^{B}. Line 13 updates the values of *CB* and L_s. This procedure is repeated until the outermost *while* loop terminates. At the end of this procedure, the algorithm selects the recharging stations that have the minimum recharging time and that are also near the points where the EV is approaching its traveling limit. After initialization, the functions *form_path()* and *selectStations()* are also used in the allocation and mutation operations to find new sub-paths between any two nodes and select recharging stations.

```
Input: $q_k.B_s$: Battery level at the source node, $q_k.T_l$: Travelling limit of the EV, $P^A$ (should be not null)
Output: $P^B$
1:  $P^B$= null, $CB = q_k.B_s \times q_k.T_l$, $L_s = 0$, $a^u = null$
2:  $L_p = \sum_{e_x \in P_A} e_x.l$
3:  while $CB < L_p$ do
4:      $P^s = null, d^s = null, a^s = null, d = 0$
5:      for each element $e_x \in P^A$ do
6:          $d = d + e_x.l$
7:          if $L_s < d \leq CB$ then
8:              $n_x = e_x.e_n$, $avg = n_x.R_T + \frac{1}{d}$
9:              $P^s = P^s \cup n_x$, $d^s = d^s \cup d$, $a^s = a^s \cup avg$
10:         end if
11:     end for
12:     $I_{sel}$ = index of the minimum value in $a^s$, $V_{sel} = d^s[I_{sel}]$, $P^B = P^B \cup P^s[I_{sel}]$
13:     $CB = V_{sel} + q_k.T_l$, $L_s = V_{sel}$
14: end while
15: return $P^B$
```

Fig. 5 Function to select recharging stations, *selectStations(P^A)*

```
Input: Any edge $e_i \in P^A$, $P_j = \{P^A, P^B\}$: A solution from the population
Output: $g(e_i) = \{g_0, g_1, g_2\}$: goodness of the edge $e_i$
1: $P^A_\alpha = P^A - e_i$, $P^B_\alpha = P^B - e_i$
2: $\delta_1 = \frac{f_1(P^B_\alpha)}{f_1(P^B)}$, $\delta_2 = \frac{f_2(P^A_\alpha)}{f_2(P^A)}$, $\delta_3 = \frac{f_3(P^A_\alpha)}{f_3(P^A)}$
3: $g(e_i) = \{\delta_1, \delta_2, \delta_3\}$
4: return $g(e_i)$
```

Fig. 6 Proposed goodness calculation

4.2 Evaluation

The evaluation step determines the goodness values of the set of edges in each solution. Figure 6 shows the method for finding the goodness of any edge e_i (where $e_i \in P^A$ and $P^A \in P_j$). The sets P^A_α and P^B_α contain all elements of the sets P^A and P^B except e_i. The goodness value has three components with values lying in the range [0,1].

4.3 Selection

For each solution, the selection operation aims to select some elements of low goodness values into its selection set. The selected elements will be later used in the allocation operation. The selection operation is applied independently to each solution. For each solution P_j, a set S_j that contains three elements (corresponding to the three objectives) is produced. Each element $i_k \in S_j$, $0 \leq k \leq 2$ is the index of the edge that has minimum value for the kth goodness objective in the solution S_j. Figure 7 shows the proposed selection operation.

```
Input: $P_j = \{P^A, P^B\}$
Output: selection set $S_j$
1:  $S_j$= null, $m_0 = \infty, m_1 = \infty, m_2 = \infty, i_0 = 0, i_1 = 0, i_2 = 0$
2:  for each element $e_j \in P^A$ do
3:      if $g(e_j)[0] < m_0$ then
4:          $m_0 = g(e_j)[0]$, $i_0 = e_j$
5:      end if
6:      if $g(e_j)[1] < m_1$ then
7:          $m_1 = g(e_j)[1]$, $i_1 = e_j$
8:      end if
9:      if $g(e_j).g_2 < m_2$ then
10:         $m_2 = g(e_j)[2]$, $i_2 = e_j$
11:     end if
12: end for
13: $S_j = \{i_0, i_1, i_2\}$
14: return S
```

Fig. 7 Proposed selection operation

```
Input: $S_j$: Selection set of the solution $P_j$, $P_j \in POP$, $d$: destination node
Output: $P_j$ (which is the solution after the allocation operation)
1:  for each edge $e_j \in S$ do
2:      $n_x = e_j.s_t$
3:      $P_\alpha = \{P^A_\alpha, P^B_\alpha\}$= $null$, $P^A_\alpha = form\_path(n_x, d)$
4:      $P^A_\alpha = concatenate\{P^A(p_0, ..., p_x), P^A_\alpha\}$, (where $p_x.e_n = n_x$)
5:      $P^B_\alpha = SelectStations(P^A_\alpha)$
6:      if $P_j.ParetoOptimal$ then
7:          if $P_\alpha \succ P_j$ then
8:              $P_j = P_\alpha$
9:          end if
10:     else
11:         if $f_1(P^B_\alpha) < f_1(P^B)$ or $f_2(P^A_\alpha) < f_2(P^A)$ or $f_3(P^A_\alpha) < f_3(P^A)$ then
12:             $P^A = P^A_\alpha$, $P^B = P^B_\alpha$
13:             $Exit$ from the $for$ loop
14:         end if
15:     end if
16: end for
17: return $P_j = \{P^A, P^B\}$
```

Fig. 8 Proposed allocation operation

4.4 Allocation operation

For each solution, the allocation operation performs mutation by altering the elements in the selection set. The allocation operation is applied to each solution in the population with probability $1{-}M_b$. Figure 8 shows the allocation operation for a solution P_j. The operation consists of a *for* loop that selects a different edge from the set S_j in each iteration. Line 2 sets n_x to be the starting node of the currently selected edge. Line 3 initializes P_α and sets P^A_α to contain a path between the nodes n_x and d. Line 4

```
Input: $P_j = \{P^A, P^B\}$, $d$ destination node
Output: $P_j$ (which is the path after the mutation operation)
1: $cnt$= 0, $P_\alpha = \{P^A_\alpha, P^B_\alpha\}$= $null$
2: for each edge $e_j \in P^A$ do
3:     if $e_j$ is not null then
4:         $cnt++$
5:     end if
6: end for
7: $r_n$= random number in the range $\{x \in Z | 0 \le x < cnt\}$
8: $n_x = e_{r_n}.s_t$
9: $P^A_\alpha = form\_path(n_x, d)$
10: $P^A_\alpha = concatenate\{P^A(p_0, ..., p_x), P^A_\alpha\}$, (where $p_x.e_n = n_x$)
11: $P^B_\alpha = SelectStations(P^A_\alpha)$
12: if $P_j.ParetoOptimal$ then
13:     if $P_\alpha \succ P_j$ then
14:         $P_j = P_\alpha$
15:     end if
16: else
17:     if $f_1(P^B_\alpha) < f_1(P^B)$ or $f_2(P^A_\alpha) < f_2(P^A)$ or $f_3(P^A_\alpha) < f_3(P^A)$ then
18:         $P^A = P^A_\alpha$, $P^B = P^B_\alpha$
19:     end if
20: end if
21: return $P_j = \{P^A, P^B\}$
```

Fig. 9 Proposed mutation Operation

forms $P^{\mathbf{A}}_\alpha$ by combining the upper portion of $P^{\mathbf{A}}$ (where $P^{\mathbf{A}} \in P_j$) with the lower portion of $P^{\mathbf{A}}_\alpha$. If P_j is a Pareto-optimal solution, then P_j is updated to P_α only if P_α dominates P_j. If P_j is not a Pareto-optimal solution then P_j is updated to P_α if P_α is better than P_j for any objective function value.

4.5 Mutation operation

The mutation operation causes random changes in solutions. In our algorithm, it is used to escape local minima. The mutation operation is applied with probability M_b to each solution in the population. The proposed mutation operation is shown in Fig. 9. Line 1 initializes the variables *cnt* and P_α. The *for* loop between lines 2 and 6 sets the value of *cnt* to the number of non-null elements in the solution $P^{\mathbf{A}} \in P_j$. Line 7 sets r_n to be a random integer between 0 and *cnt-1*. n_x is the starting node of the edge that lies at position r_n in P_A of P_j. Using n_x, a new path P_α is formed with the function *form_path()* that is also used in the allocation operation. P_j is updated to P_α under different conditions that are based on whether P_j is a Pareto-optimal solution.

4.6 Mark Pareto-optimal solutions

In this step, the solutions in the population that are Pareto-optimal are marked by setting the value of their attribute *ParetoOptimal* to true.

5 Estimation of memory requirements

Now, we estimate the memory required to store the number of paths generated by our proposed algorithm. The number of solutions stored by the algorithm is equal to the population size N. The algorithm also creates an additional path (P_α) in the allocation and mutation operations. Therefore, the total number of paths that are stored in the memory is $N + 1$. If we assume that one solution consumes δ units of memory, our proposed algorithm requires $(N + 1)\delta$ units of memory. For comparison purposes, we also estimate the memory required by the NSGA-II. The number of paths stored by NSGA-II is twice its population size because the number of children it creates is equal to its population size. Therefore, if N' is the population size in the NSGA-II, $2N'\delta$ units of memory are required. The ratio of the number of paths created by NSGA-II to that created by our proposed algorithm is $\frac{\text{mem}_{\text{NSGA-II}}}{\text{mem}_{\text{proposed}}} = \frac{2N'}{N+1}$. This reduction of memory usage is effective for the implementation on an embedded system with insufficient resources.

6 Simulations

We implemented our proposed algorithm and NSGA-II in Visual Studio C# and ran them on an Intel Celeron 2-MHz laptop computer with 2 GB RAM. The Celeron is one of Intel's low-cost processors that are also used in embedded systems in EVs. The rest of this section describes the details of the experiments and analyzes the results.

The experiments used the San Francisco Bay Area (*BAY*) and Colorado (*COL*) road networks [20]. *BAY* contains 321,250 nodes and 800,172 edges and *COL* contains 435,666 nodes and 1,057,066 edges. The road networks contain information about the starting and ending nodes of the edges ($e_j \times s_t$ and $e_j \times e_n$), the length of the edges ($e_j \times l$), and the average traveling time on the edges ($e_j \times S_T$). Therefore, the values of four out of the seven properties described in Table 1 are based on road network information. The values of properties related to the nodes and EVs were determined as follows. Recharging stations were assumed to be present at all nodes. Recharging stations that can support rapid recharging have recharging times as short as 10 min [2]; however, some rapidly recharging stations have recharging times as long as 30 min [21]. Therefore, the recharging times at the nodes ($n_i \times R_T$) are assumed to lie between 10 and 30 min. The traveling limit of the EVs ($q_k \times T_l$) is generally 120 km [22]. The last property, the battery level at the source node, was assumed to randomly vary between 60 and 100 %. Assume that the geographical locations of the source (s) and destination (d) nodes are represented as $s(x_s, y_s)$ and $d(x_d, y_d)$, respectively. The distance (d) between s and d

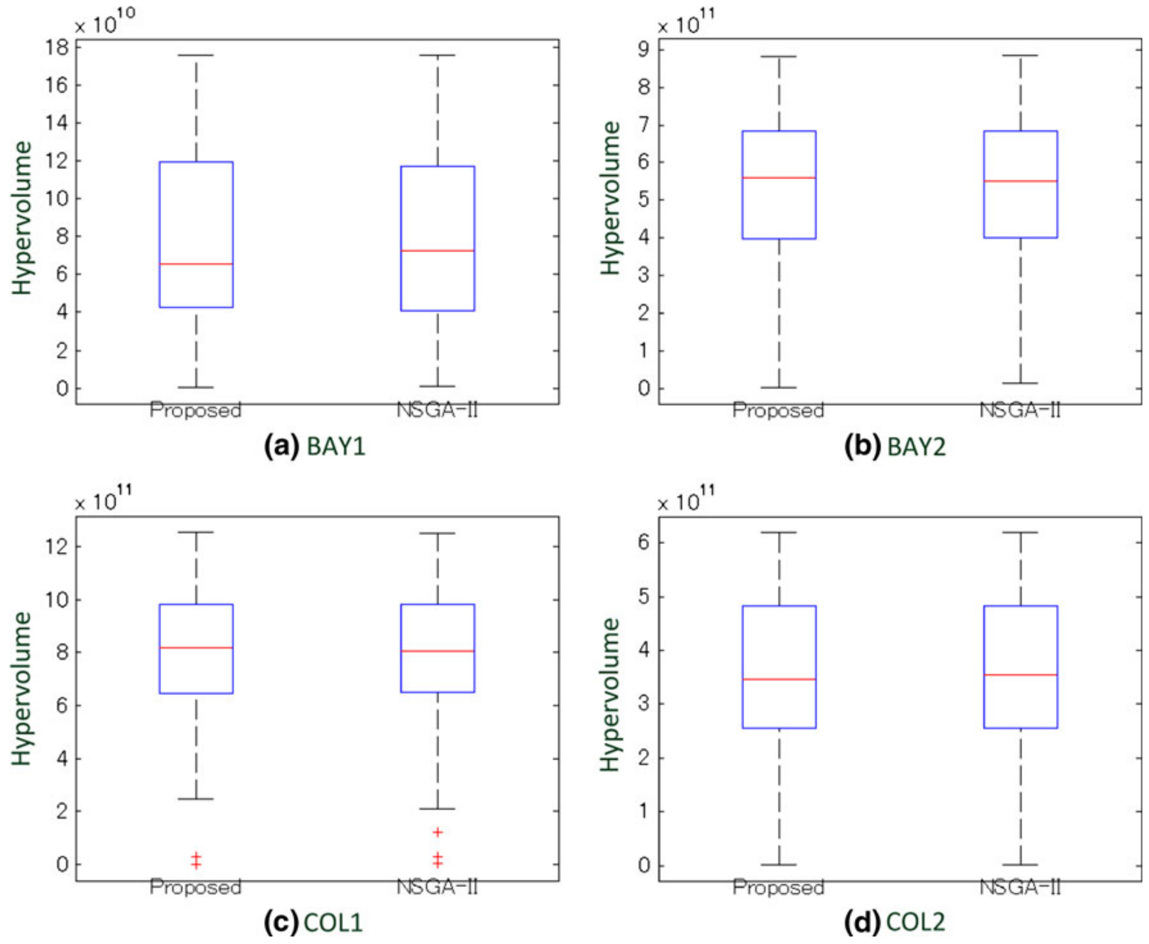

Fig. 10 Box-and-whisker plots

Table 2 Results of the Wilcoxon rank sum tests

Experiments	*P* values
BAY1	0.94
BAY2	1.00
COL1	0.90
COL2	0.87

nodes can be determined using the distance formula, i.e., $d = \sqrt{(x_d - x_s)^2 + (y_d - y_s)^2}$. Each experiment involved randomly selecting s and d nodes, such that, $100\,\text{km} < d < 300\,\text{km}$ and using our proposed algorithm and NSGA-II to find Pareto-optimal paths between them. In road networks, many obstacles exist between s and d nodes, therefore, shortest path between them is generally much longer than d. Many US cities have total area around 300 km^2 [23], therefore, the EV can travel within a small city in the selected range of d. The experiments were performed under the following four conditions: (1) in *BAY1*, up to 100 experiments were executed on the *BAY* road network with $N = 10$, $N' = 10$ (recall that N is the population size for our proposed algorithm and N' is the population size for NSGA-II), (2) in *BAY2*, up to 100 experiments were executed on the *BAY* road network with $N = 20$, $N' = 10$, (3) in *COL1*, up to 100 experiments were executed on the *COL* road network with $N = 10$, $N' = 10$, (4) in *COL2*, up to 100 experiments were executed on the *COL* road network with $N = 20$ and $N' = 10$. When $N = 10$ and $N' = 10$, $\frac{\text{mem}_{\text{NSGA-II}}}{\text{mem}_{\text{proposed}}} = 1.82$, and when $N = 20$ and $N' = 10$, $\frac{\text{mem}_{\text{NSGA-II}}}{\text{mem}_{\text{proposed}}} = 0.95$.

Therefore, for the experiments in *BAY1* and *COL1*, our proposed algorithm requires approximately half amount of memory required by NSGA-II, and for the experiments in *BAY2* and in *COL2*, our proposed algorithm and NSGA-II require the same amount of memory. The values of the remaining parameters of the algorithms were as follows. M_b, which is the mutation probability, was set to 0.15. The crossover and mutation probabilities in NSGA-II were set to 1 and 0.15, and the algorithm selected parents for the crossover operation using tournament selection based on

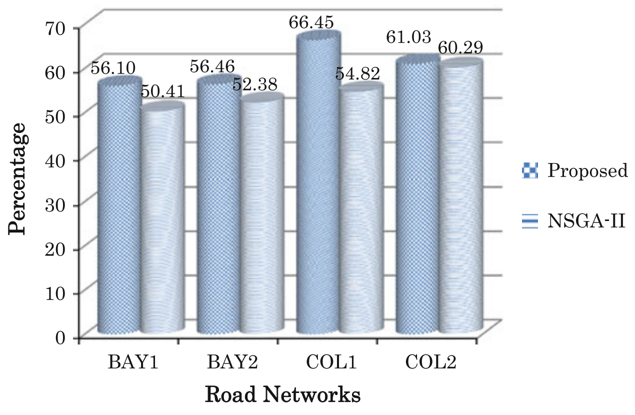

Fig. 11 Contribution of the algorithms to the overall Pareto-optimal solutions

crowding distance. The crossover and mutation operations proposed by Ahn [24] were used. The stopping criterion in all experiments was set to an execution time of 30 s.

The hypervolume of the Pareto-optimal solutions found for the experiments was determined using the hypervolume calculation tool proposed by Carlos et al. [25], and the bounding point in the hypervolume calculation was determined using the method proposed by Knowles [26]. The results of the hypervolume calculations in the experiments are shown with box-and-whisker plots in Fig. 10. In addition, we used Wilcoxon rank sum test [27] to compare the hypervolumes of the Pareto-optimal solutions obtained by the two algorithms in the different experiments. The tests were applied with the significance level $\alpha = 0.05$. The rank sum tests yield P values and when the P values are less than or equal to α, the hypervolume distributions of the two algorithms are not significantly different from each other. The results of the rank sum tests are shown in Table 2. The P values in all test cases are greater than $\alpha = 0.05$, and therefore, the difference between the hypervolumes obtained from the proposed algorithm and NSGA-II is insignificant.

In the experimentation, it was noted that an increase in the execution time to 120 s can allow one to double the population size and increase the upper bound on the value of d to $+\infty$.

In addition, the performance of the algorithms was measured by determining their share in the overall Pareto-optimal solutions. The algorithm that has a higher share is better, and this can be determined by the following method. During any experiment, the results of the two algorithms were combined and an overall Pareto-optimal set was calculated. The number of solutions found by an algorithm that is member of the overall Pareto-optimal set determines its share in the overall Pareto-optimal set. The results, which are displayed with bar graphs in Fig. 11, show that the performance of our proposed algorithm was better than that of NSGA-II in most of the test cases. On an average, our proposed algorithm found 5.5 % more solutions in the overall Pareto-optimal sets than NSGA-II.

7 Conclusion

This study solves the OPS problem for EVs. We assumed that road networks have recharging stations located at their nodes. A population-based SimE algorithm was used to perform OPS. Our proposed algorithm has innovative functions for calculating goodness values and performing the SimE allocation operation. The algorithm stores $N + 1$ paths in the memory, where N is the population size. The performance of our proposed algorithm was compared with that of NSGA-II both running on an Intel Celeron processor, which can also be used in embedded systems in EVs. Our proposed algorithm found an average of 5.5 % more Pareto-optimal solutions than NSGA-II. Therefore, the experimental results show that our proposed algorithm is suitable for implementation on the embedded systems used in EVs.

References

1. Saber AY, Venayagamoorthy GK (2009) One million plug-in electric vehicles on the road by 2015. In: International IEEE Conference on Intelligent Transportation Systems, St. Louis
2. Amoli ME, Choma K, Stefani J (2010) Rapid-charge electric-vehicle stations. IEEE Trans Power Delivery 25(3):1883–1887
3. Li Z, Sahinoglu Z, Tao Z, Teo KH (2010) Electric vehicles network with nomadic portable charging stations. In: IEEE 72nd Conference on Vehicular Technology Fall, VTC-2010 Fall, Ottawa
4. Sundstorm O, Binding C (2010) Planning electric-drive vehicle charging under constrained grid conditions. In: Internal Conference on Power System Technology (POWERCON), Zhejiang
5. Strom E, Hartenstein H, Santi P, Wielsbeck W (2010) Vehicular communications: ubiquitous networks for sustainable mobility. Proc IEEE 98(7):1111–1112
6. Tarapata Z (2007) Selected multicriteria shortest path problems: an analysis of complexity, models and adoption of standard algorithms. Int J Appl Math Comput Sci 17(2):269–287
7. Garey MR, Johnson DS (1997) Computers and intractability: a guide to the theory of NP-completeness. W. H Freeman, New York
8. Sait SM, Youssef H (1999) Iterative computer algorithms with applications in engineering. IEEE Computer Society Press, New York
9. Siddiqi UF, Shiraishi Y, Sait SM (2011) Multi constrained route optimization for electric vehicles using SimE. In: International Conference on Soft Computing and Pattern Recognition (SoCPaR), Dalian
10. Song Q, Wang X (2011) Efficient routing on large road networks using hierarchical communities. IEEE Trans Intell Trans Systems 12(1):132–140
11. Martins E, Santos J (1999) The labeling algorithm for the multiobjective shortest path problem, Departamento de Matematica, Universidade de Coimbra, TR-99/005, Portugal

12. dos Santos Coelho PAL (2008) Multiobjective electromagnetic optimization based on a nondominated sorting genetic approach with a chaotic crossover operator. IEEE Trans Magnetics 44(6): 1078–1081
13. Yinhong L, Xianzhong D, Zhihuan L (2010) Non-dominated sorting genetic algorithm-ii for robust multi-objective optimal reactive power dispatch. IET Generation Trans Distribution 4(9):1000–1008
14. Chandrasekaran K, Kandaswamy A, Rio G, D'Souza L (2010) Improved NSGA-II based on a novel ranking scheme. J Comput 2(2):91–95
15. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 2(6):182–197
16. Lebensztajn L, Coelho LDS, Bora TC (2012) Non-dominated sorting genetic algorithm based on reinforcement learning to optimization of broad-band reflector antennas satellite. IEEE Trans Magnetics 48(2):767–770
17. Li X (2003) A nondominated sorting particle swarm optimizer for multiobjective optimization. In: International Conference on Genetic and Evolutionary Computation (GECCO 2003) Part I, Chicago
18. Villez K, Gupta A, Venkatasubramanian V (2011) Resilient design of recharging station networks for electric transportation vehicles. In: Fourth International Symposium on Resilient Control Systems (ISRCS), Boise
19. Sait SM, Ali MI, Zaidi AM (2007) Evaluating parallel simulated evolution strategies for vlsi cell placement. J Math Model Algorithms 6(3):433–454
20. 9th DIMACS Implementation Challenge—Shortest Paths [Online]. http://www.dis.uniroma1.it/challenge9/download.shtml
21. North Carolina Advanced Energy Corporation (2011) Charging station installation handbook
22. Kiyama N, Aoshima H, Kashiyama M, Kobayashi Y (2011) A route search method for electric vehicles in consideration of range and locations of charging stations. In: IEEE Intelligent Vehicles Symposium (IV), Baden
23. List of United States cities by area [Online]. http://en.wikipedia.org/wiki/List_of_United_States_cities_by_area
24. Ramakrishna RS, Ahn CW (2002) A genetic algorithm for shortest path routing problem and the sizing of populations. IEEE Trans Evol Comput 6(6):566–579
25. Fonseca MC, Paquete L, Paquete ML (2006) An improved dimension–sweep algorithm for the hypervolume indicator. In: 2006 IEEE Congress on Evolutionary Computation (CEC'06), Piscataway
26. Knowles J (2005) ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problem. IEEE Trans Evol Comput 10(1):50–66
27. Hollander M, Wolfe DA (1999) Nonparametric statistical methods. Wiley, New York