# Evolutionary Algorithms for Designing Self-sufficient Floating Neighborhoods

Kirimtat, Ayca; Ekici, Berk; Çubukçuoglu, Cemre; Sariyildiz, Sevil; Tasgetiren, Mehmet Fatih

**DOI**
10.1007/978-3-030-01641-8_6

**Publication date**
2019

**Document Version**
Final published version

**Published in**
Optimization in Industry

**Citation (APA)**
Kirimtat, A., Ekici, B., Çubukçuoglu, C., Sariyildiz, S., & Tasgetiren, M. F. (2019). Evolutionary Algorithms for Designing Self-sufficient Floating Neighborhoods. In S. Datta, & J. Davim (Eds.), *Optimization in Industry: Present Practices and Future Scopes* (pp. 121-147). Springer. https://doi.org/10.1007/978-3-030-01641-8_6

# Evolutionary Algorithms for Designing Self-sufficient Floating Neighborhoods


**Ayca Kirimtat, Berk Ekici, Cemre Cubukcuoglu, Sevil Sariyildiz and Fatih Tasgetiren**

**Abstract** Floating neighborhoods are innovative and promising urban areas for challenges in the development of cities and settlements. However, this design task requires a lot of considerations and technical challenges. Computational tools and methods can be beneficial to tackle the complexity of floating neighborhood design. This paper considers the design of a self-sufficient floating neighborhood by using computational intelligence techniques. In this respect, we consider a design problem for locating each neighborhood function in each cluster with a certain density within a floating neighborhood. In order to develop a self-sufficient floating neighborhood, we propose multi-objective evolutionary algorithms, namely, a self-adaptive real-coded genetic algorithm (CGA) as well as a self-adaptive real-coded genetic algorithm (CGA_DE) employing mutation operator of differential evolution algorithm. The only difference between CGA and CGA_DE is the fact that CGA uses random immigration of certain individuals into the population as a mutation operator whereas in the mutation phase of CGA_DE algorithm, the traditional mutation operator DE/rand/1/bin of DE algorithms. The arrangement of individual functions to develop each neighborhood function is further elaborated and formed by using Voronoi diagram algorithm. An application to design a self-sufficient floating neighborhood in Urla district, which is on the west coast of Turkey, İzmir, is presented.

A. Kirimtat · B. Ekici
Department of Architecture, Yasar University, İzmir, Turkey
e-mail: ayca.kirimtat@yasar.edu.tr

B. Ekici
e-mail: berk.ekici@yasar.edu.tr; B.Ekici-1@tudelft.nl

C. Cubukcuoglu
Department of Interior Architecture and Environmental Design, Yasar University, İzmir, Turkey
e-mail: cemre.cubukcuoglu@yasar.edu.tr; C.Cubukcuoglu@tudelft.nl

B. Ekici · C. Cubukcuoglu · S. Sariyildiz
Chair of Design Informatics, TU Delft, Delft, The Netherlands
e-mail: I.S.Sariyildiz@tudelft.nl

F. Tasgetiren (✉)
Department of International Logistics and Management, Yasar University, İzmir, Turkey
e-mail: fatih.tasgetiren@yasar.edu.tr



S. Datta and J. P. Davim (eds.), *Optimization in Industry*, Management and Industrial Engineering, https://doi.org/10.1007/978-3-030-01641-8_6

**Keywords** Computational design · Performance-based design · Self-sufficient Floating city · Multi-objective optimization · Urban design · Genetic algorithm Differential evolution · Form-finding · Voronoi diagram

## 1 Introduction

In the past decades, relevant and innovative solutions for cities have been exploring among researchers, city planners, as well as engineers. The reason is to tackle environmental problems caused by rising sea levels, natural disasters, and harmful effects of human activities. It is predicted by researchers that sea levels will continue to rise around the world [1]. For these reasons, we will see to face the problem of land shortage in the near future. In addition, extraordinary natural events such as earthquakes and heat waves will occur more frequent and more intense in the future. Many of the vulnerable cities, which are often in coastal locations, can be unpleasantly affected by problems mentioned above. In addition, a large number of world population is living in the coastal areas. For this reason, we have to consider the challenges in the development of cities and settlements to save humanity and our future life. Some of European and East Asian cities such as Tokyo, Shanghai, London, Rotterdam, and Hamburg take precautions against those challenges [2]. However, more advanced design solutions are still being sought in the world. Regarding this, floating settlements have emerged as new promising urban areas. Since the oceans are technically seen as international zones, they are defined as our last chance to remain in life on the earth [3]. In opposition to the traditional land renovation methods, floating settlements provide an exciting and environmentally friendly solution for land creation [3]. They have many advantages in terms of various aspects stated as follows:

- Protecting the marine eco-system,
- Construction on the sea is fast and easy,
- Easily removed or expanded,
- Durable against the seismic shocks,
- Presenting economical solutions especially when the sea depth is high or the seabed is very soft [4].

There are relatively few examples of floating settlements, such as

- A collaborative design of The SeaSteading Institute with Delta Sync [3],
- Floating city design proposal of AT Design Office in China [5],
- A concept design proposed by Baca Architects [6].

In the examples mentioned above, most of them deal with structural aspects, energy-efficiency, self-sufficiency, growth, movability, seakeeping, safety, water experience, and cost-efficiency. Based on these approaches, we aimed to consider both architectural and engineering design goals. We foresee that combining these two disciplines in the conceptual phase can play a key role in the development of future floating settlements.


**Fig. 1** Project region: Urla, İzmir, Turkey

As an extension of [7], this paper develops a novel design methodology for the design of a self-sufficient floating neighborhood through implementations of evolutionary algorithms. Briefly, we combine knowledge from Floating Settlements Design practice with Computational Intelligence. Regarding the formulation of the complex problem in our study, we start with clustering the proposed site region into different zones. Then, we focus on locating each neighborhood function in each cluster with certain percentages. The neighborhood functions are residential, agricultural, public, and green areas. We aim to fit 30.000 people in this floating neighborhood. In the problem of distribution of the neighborhood functions in each cluster, inputs are the percentages of each function in each cluster and the cluster capacities. On the other hand, objective functions are to ensure walkability, scenery of the residents, as well as to enable the design to be cost-effective. We aim at finding desirable distributions of the functions in each cluster that contribute the self-sufficient design of floating neighborhoods. To solve this complex problem, we applied two different evolutionary algorithms, namely CGA and CGA_DE. The arrangement of individual functions to develop each neighborhood function is further elaborated and formed by using Voronoi diagram algorithm. An application to design a self-sufficient floating neighborhood in Urla, which is a coastal region located on the west of Turkey, İzmir is presented. So-called project region can be found in Fig. 1.

The rest of the paper is organized as follows: Sect. 2 introduces the design methodology and problem formulation developed in this chapter. Section 3 presents the evolutionary algorithms that have been applied in this chapter. Section 4 discusses

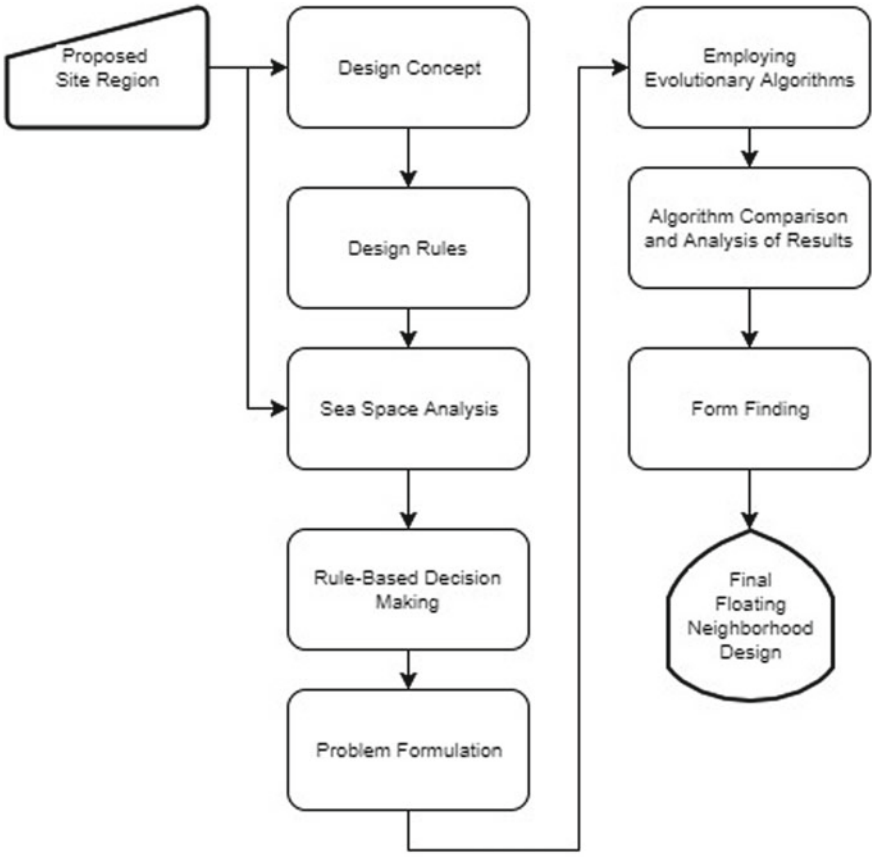


**Fig. 2** Floating neighborhood design workflow

the computational results. Section 5 presents the form-finding step. Finally, Sect. 6 gives the conclusions.

## 2 Design Methodology

In this section, we introduce our proposed self-sufficient floating neighborhood design and its fundamentals. The design process consists of several steps as shown in Fig. 2.

## *2.1 Design Concept*

The design concept is in relation with proposed project region. According to our preliminary site analysis, the urban area is required for further investigation in the conceptual phase. Thus, the design concept for the floating neighborhoods involves the following features:

- Proposed design should be **self-sufficient**. The food will be provided from the neighborhoods themselves by the help of agricultural areas.
- Distances between each function should be **walkable**. There will be fewer vehicles for transporting from one function to another.
- **Scenic** view should be desirable for both the residents in the project region and in the proposed floating neighborhood.
- The construction on the sea should be **cost-efficient** and required low budgets.

## *2.2 Design Rules*

We infer the design rules through "if-then" statements, which is a common approach for representing design knowledge. In this approach, the design inferences are based on a process of obtaining new knowledge through existing knowledge. We could rather define them as our propositions, too. The design rules are mainly related to seabed characteristics of the project region, construction budget for residential areas, distances among the neighborhood functions and some specific proximity requirements. "if-then" rules considering in this study are listed as follows:

- If (sea depth is very shallow), then it is not proper for any neighborhood.
- If (sea depth is shallow) and (budget for residential function is low), then total budget is cost-effective.
- If (agricultural functions are very close to each other), and (residential functions are far from each other), then it is scenic for Urla people.
- If (green area functions are very close to each other), and (residential functions are far from each other), then it is scenic for Urla people.
- If there is a (very short distance between public and residential functions), and (very short distance between residential function and agricultural function), then it is a walkable city.
- If there is a (very short distance between residential function and public function), and (very short distance between residential function and green function), then it is a walkable city.

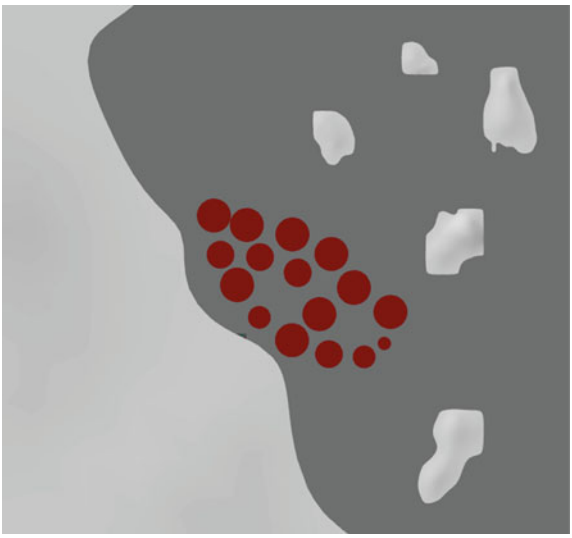

**Fig. 3** Placements of 16 clusters on the intervention area

## *2.3 Sea Space Analysis*

Sea space is analyzed to determine the proper intervention area for the location of the floating neighborhood. This decision is related to the water depth information. We selected the intervention area along the coastline that has water depths in between 5 and 20 m. Selected area is also based on our "if-then" rules stated above. As shown in Fig. 3, we divided the intervention area into 16 zones, so-called clusters where we distribute each neighborhood function in them.

## *2.4 Rule-Based Decision-Making*

Rule-based decision-making is a way to determine which neighborhood function will be placed in which cluster with how much density. In other words, the decisions regarding the distribution of the neighborhood functions are based on "if-then" rules considering design goals, which are scenery, walkability, and cost-effectiveness. For instance, one cluster can separate its functions as 35% of green areas, 45% of residential areas, 20% of agricultural areas, and 0% of public areas. On the other hand, another cluster's percentages could be divided as 0% of green areas, 20% of residential areas, 40% of agricultural areas, and 40% of public areas. These percentages directly affect the performance of the design alternatives.

## 2.5 Problem Formulation

In this section, we explain the mathematical model of our proposed floating neighborhood design problem. Notations of decision variables, objectives, and constraints are given in Table 1.

According to the notations above, the mathematical model is described as

$$\begin{aligned} &\min\left(\tfrac{1}{Ec}, \tfrac{1}{Sc}, \tfrac{1}{Wa}\right) \\ &\text{subject to} \\ &100,000\,\text{m}^2 < aa < 120.000\,\text{m}^2 \end{aligned} \tag{1}$$

$$0 < p_{xy} < 1 \tag{2}$$

$$ra \geq 300,000\,\text{m}^2 \tag{3}$$

$$ca \geq 900,000\,\text{m}^2 \tag{4}$$

***Decision variables*** of the floating neighborhood design problem consist of 64 variables, which are percentages to distribute four functions into 16 clusters.

***Cost-Effectiveness,*** denoted as $Ec$, is one of the problem objectives as given in Eqs. (5)–(9). There is a close relationship between water depths and budget requirements. Locating the neighborhoods on shallow sea depths is an economical solution for settlements on the sea. Due to the cost-effectiveness priority, clusters are forced to locate on sea depth with (0–5 m) or (5–20 m). On the other hand, this objective aims at keeping total budget of the floating neighborhood between 1 million TL and 1.5 million TL for residential areas because of the highest budget requirements of them. These cost values are measured based on the proposed number of 30,000 people living in the neighborhood.

$$ss = \max\left(0, \min\left(1, \frac{sd - 20}{5 - 20}\right)\right) \tag{5}$$

$$su = \max\left(0, \min\left(1, \frac{sd - 5}{0 - 5}\right)\right) \tag{6}$$

$$abl = \max\left(0, \min\left(1, \frac{db - 1.5}{1 - 1.5}\right)\right) \tag{7}$$

where

$$db = ra * cu \tag{8}$$

$$Max\ Ec = ss + su + abl \tag{9}$$

***Scenery*** objective consists of two main criteria: scenery for the residents in the floating neighborhood ($Scf$) and scenery from Urla to floating neighborhood ($Scu$). In order to achieve $Scf$ part of this objective, we aim at keeping all functions close to the agricultural and the green areas to enable more sea view for residents in the

**Table 1** Problem notations

| Notations | Descriptions |
|---|---|
| $p_{xy}$ | Density of function ($x$) in cluster ($y$) |
| $x$ | Index of functions $x = 1, \ldots, 4$ |
| $y$ | Index of clusters $y = 1, \ldots, 16$ |
| $aa$ | Capacity of each cluster |
| $sd$ | Sea depth (m) |
| $Ec$ | Cost-effectiveness |
| $Sc$ | Scenery |
| $Wa$ | Walkability |
| $ss$ | Shallow water |
| $su$ | Shallow water |
| $db$ | Total budget for residential areas |
| $cu$ | Unit cost of residential areas $\left(\frac{\text{cost}}{\text{m}^2}\right)$ |
| $ra$ | Total area of residential functions $\left(\text{m}^2\right)$ |
| $ca$ | Total area of agricultural functions $\left(\text{m}^2\right)$ |
| $abl$ | The degree of low budget satisfaction |
| $Scf$ | Scenery for residents in floating neighborhood |
| $Scu$ | Scenery for Urla people |
| $Ox$ | $x$ coordinate of offshore |
| $Oy$ | $y$ coordinate of offshore |
| $Cx$ | $x$ coordinate for coastline of Urla |
| $Cy$ | $y$ coordinate for coastline of Urla |
| $Rx$ | $x$ coordinate of residentials |
| $Ry$ | $y$ coordinate of residentials |
| $Ax$ | $x$ coordinate of agricultural areas |
| $Ay$ | $y$ coordinate of agricultural areas |
| $Gx$ | $x$ coordinate of green areas |
| $Gy$ | $y$ coordinate of green areas |
| $Px$ | $x$ coordinate of public areas |
| $Py$ | $y$ coordinate of public areas |
| $W_{xixj}$ | Walkability between functions $xi$ and $xj$<br>1: residential, 2: agricultural, 3: green, 4: public |
| $D_{R,O}$ | Manhattan distance between residential and offshore |
| $D_{A,C}$ | Manhattan distance between agricultural and coast |
| $D_{G,C}$ | Manhattan distance between green and coast |
| $D_{R,C}$ | Manhattan distance between residential and coast |
| $D_{R,G}$ | Manhattan distance between residential and green |
| $D_{R,A}$ | Manhattan distance between residential and agricultural |
| $D_{R,P}$ | Manhattan distance between residential and public |

floating neighborhood. Regarding the *Scu* part of this objective, we aim at locating the residential areas more close to the water shore, in contrast, locating the agricultural and green areas more close to the Urla coastline. To assess the distances between functions and other places, we make use of Manhattan distance calculations as shown in Eqs. (10)–(13). After the distances are calculated, we aim at keeping those distance values in a certain interval as stated in below equation from (14) to (17). This interval is in between 300 and 1500 m for $D_{A,C}, D_{G,C}, D_{R,C}$. On the other hand, the acceptable range for the distance between residential and offshore is 0–1500 m.

$$D_{R,O} = |Rx - Ox| + |Ry - Oy| \tag{10}$$

$$D_{A,C} = |Ax - Cx| + |Ay - Cy| \tag{11}$$

$$D_{G,C} = |Gx - Cx| + |Gy - Cy| \tag{12}$$

$$D_{R,C} = |Rx - Cx| + |Ry - Cy| \tag{13}$$

$$0 < D_{R,O} < 1500 \tag{14}$$

$$300 < D_{A,C} < 1500 \tag{15}$$

$$300 < D_{G,C} < 1500 \tag{16}$$

$$300 < D_{R,C} < 1500 \tag{17}$$

Mathematical expressions that aim at keeping the distances between ranges are given from Eqs. (18)–(22).

$$d_1 = \max\left(0, \min\left(1, \frac{D_{R,O} - 1500}{0 - 1500}\right)\right) \tag{18}$$

$$d_2 = \max\left(0, \min\left(1, \frac{D_{A,C} - 1500}{300 - 1500}\right)\right) \tag{19}$$

$$d_3 = \max\left(0, \min\left(1, \frac{D_{G,C} - 1500}{300 - 1500}\right)\right) \tag{20}$$

$$d_4 = \max\left(0, \min\left(1, \frac{D_{R,C} - 1500}{300 - 1500}\right)\right) \tag{21}$$

$$\max Sc = \left(\frac{1}{min(d_1, (\min(\max(d_2, d_3), d_4))}\right) \tag{22}$$

***Walkability*** is another objective of this design problem. Numerical walkability scores that assigned to each location according to their proximity related features are gathered from walk score. Walk score website [8] searches for walkable cities based on easy access to public transit, better commutes, and proximity to the people and places you love. Based on this approach, equation from (23) to (27) show the calculations of distances between functions. Then, we try to keep those distances in a walkable range.

$$D_{R,G} = |Rx - Gx| + |Ry - Gy| \tag{23}$$

$$D_{R,A} = |Rx - Ax| + |Ry - Ay| \tag{24}$$

$$D_{R,P} = |Rx - Px| + |Ry - Py| \tag{25}$$

$$300 < D_{xi,xi} < 1000 \quad i = 1, \ldots, 4 \; and \; j = 1, \ldots, 4 \tag{26}$$

$$\text{maximize } W_{xi,xj} = \max\left(0, \min\left(1, \frac{D_{xi,xj} - 1000}{300 - 1000}\right)\right) \quad for \; \forall \, xi \; and \; xj. \tag{27}$$

***Problem Constraints*** are categorized into three parts as "Cluster Capacity Constraint", "Black Box Constraint" and "Functions' Areas Constraint". Cluster capacity constraint is given in Eq. (1), black box constraint is given in Eq. (2) and area constraints for neighborhood functions are given in Eqs. (3) and (4).

## 3 Proposed Evolutionary Algorithms

In this chapter, we present multi-objective evolutionary algorithms to solve the floating settlement design problem. Evolutionary algorithms (EAs) are popular optimization algorithms since they have been implemented to multi-objective problems (MOP). For this reason, they are entitled as multi-objective evolutionary algorithms (MOEA) . Amongst them, NSGA-II [9] and SPEA-2 [10] are the most studied ones to deal with MOPs in the literature. As an extension of [7], CGA and CGA_DE are developed and implemented in order to solve the complex floating neighborhood design problem.

Genetic algorithms (GA) are search heuristics based on the biological process of natural selection and evolution [11]. In GAs, individuals with decision variables in $D$ dimensions are encoded into chromosomes to obtain an initial population that should be evolved over generations. At each generation, two individuals are chosen and mated from the population. Then, two individuals are crossed over to generate new solutions called offspring or child. Some individuals are mutated to escape from local minima. Ultimately, offspring population is added to parent population in order to select new individuals for the next generation. Figure 4 shows the overall scheme of the genetic algorithm (GA).

Real-coded GA needs a crossover and mutation operators. As a crossover operator, we employ a simple binomial crossover operator to generate offspring $Q_i^t$. In other words, two individuals, $X_a$ and $X_b$, are selected from the parent population. Then, each dimension of the offspring is either taken from the first or second individual with a certain crossover probability $CR_i^t$. The outline of the crossover operator is given in Fig. 5 to generate offspring $Q_i$ as well as an example is given in Table 2.

Regarding the mutation operator, some dimensions of offspring $Q_i^{j,t+1}$ can be mutated or perturbed with a small mutation probability as shown in Table 2.

**Fig. 4** GA procedure

$$
\begin{aligned}
&Establish\ initial\ population\ P^t\ at\ generation\ t\\
&Evaluate\ individuals\ in\ P^t\\
&While\ (not\ termination) do\\
&\{\\
&\quad Select\ two\ individuals\ from\ P^t\\
&\quad Crossover\ individuals\ to\ produce\ offspring\ Q^t\\
&\quad Mutate\ some\ individuals\ in\ Q^t\\
&\quad Add\ offspring\ Q^t\ to\ individuals\ in\ P^t\\
&\quad Evaluate\ (P^t + Q^t) individuals\ in\ P^t\\
&\quad Select\ P^t\ individuals\ from\ (P^t + Q^t)\\
&\}\\
&End\ While\\
&End\ Algorithm
\end{aligned}
$$

**Fig. 5** Crossover operator

$$
\begin{aligned}
&for\ j = 1\ to\ D\\
&\quad if\ r_j < CR_i\ then\\
&\quad\quad Q_i^{j,t+1} = X_a^{j,t}\\
&\quad else\\
&\quad\quad Q_i^{j,t+1} = X_b^{j,t}\\
&endfor
\end{aligned}
$$

**Table 2** Binomial crossover and mutation operator

| $j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $CR_i$ | 0.70 | 0.70 | 0.70 | 0.70 | 0.70 |
| $r_j$ | 0.80 | 0.25 | 0.92 | 0.67 | 0.11 |
| $x_a^{j,t}$ | 0.15 | 0.70 | 0.35 | 0.45 | 0.95 |
| $x_b^{j,t}$ | 0.65 | 0.75 | 0.10 | 0.25 | 0.05 |
| $Q_i^{j,t+1}$ | 0.65 | 0.70 | 0.10 | 0.45 | 0.95 |
| $r_j$ | 0.80 | 0.01 | 0.18 | 0.75 | 0.15 |
| $MR_i$ | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| $Q_i^{j,t+1}$ | 0.65 | 0.32 | 0.10 | 0.45 | 0.95 |

## *3.1 CGA and CGA_DE Algorithms*

The real-coded GA mentioned above is actually for single objective real-parameter optimization problems. Now, in this section, we extend it to multi-objective floating neighborhood design problem. We propose a self-adaptive real-coded CGA and CGA_DE algorithms for the problem on hand. In both algorithms, initial popula-

tion is constructed uniformly and randomly within the given boundaries for each of 64 dimensions. The binomial crossover operator is used in both algorithms to generate offspring population. The difference between two algorithms comes from the different mutation operators used. In the CGA, we immigrate some random new individuals into the parent population with an amount of $MR_i \times \left|P^t\right|$. It means that if the $MR_i$ is equal to 0.02 with the population size $\left|P^t\right| = 100$, two individuals selected from the population are randomly and uniformly regenerated within the boundaries of each dimension. On the other hand, we employ the traditional mutation operator DE/rand/1/bin of DE algorithms. In other words, a certain percent of the offspring population randomly goes under a mutation operator, after generating the offspring population. For the mutation operator, we propose a distinct strategy by employing the mutation operator of DE. Three individuals are randomly chosen from the parent population. Then, the difference between two individuals is multiplied by a uniform random number $r$ between 0 and 1 in order to add to another individual randomly chosen as in Eq. (28).

$$Q_i^{j,t+1} = X_a^{j,t} + r \times \left(X_b^{j,t} - X_c^{j,t}\right) \tag{28}$$

Self-adaptive procedure in DE algorithms is first proposed by Brest et al. [12, 13]. We employ the same idea in this paper. Initially, $CR_i$ and $MR_i$ values are assigned to 0.9 and 0.05. These values are updated for each individual I at each generation t as in as in Eqs. (29) and (30).

$$MR_i^{t+1} = \begin{cases} MR_{min} + r_1 \cdot MR_{max} & if\ r_2 < p_1 \\ MR_i^t & otherwise \end{cases} \tag{29}$$

$$CR_i^{t+1} = \begin{cases} CR_{min} + r_1 \cdot CR_{max} & if\ r_2 < p_1 \\ CR_i^t & otherwise \end{cases} \tag{30}$$

where $r_j \in \{1, 2\}$ are uniform random numbers in the range [0, 1]. $p_1$ denotes the probability to adjust the $CR_i$ and $MR_i$ values. Parameters are taken as $p_1 = 0.1$, $CR_{min} = 0.1$, and $CR_{max} = 0.9$. In addition, $MR_{min}$ and $MR_{max}$ are taken as 0.01 and 0.05.

For both algorithms, we take advantage of non-dominated sorting procedure and constrained-domination rule of the NSGA-II algorithm [9], which is one of the most sophisticated multi-objective algorithms in the literature. The non-dominated sorting procedure is given in Fig. 6.

However, we employ the fast non-dominated sorting algorithm to create non-dominated fronts in both algorithms proposed in this paper.

Another key feature of the NSGA-II algorithm is the crowding distance. Suppose that we obtained a non-dominated set $\delta$. The crowding distance of $\delta[i]_{dist}$ is calculated as in Fig. 7.

```
for each p ∈ P
    S_p = ∅
    n_p = 0
    for each q ∈ P
        if (p ≺ q) then                    If p dominated q
            S_p = S_p ∪ {q}                Add q to the set of solutions dominated by p
        else if (q ≺ p) then
            n_p = n_p + 1                  Increment the domination counter of p
    if n_p = 0 then                        p belongs to the first front
        p_rank = 1
        F_1 = F_1 ∪ {p}
i = 1                                      Initialize the front counter
while F_i ≠ ∅
    Q = ∅                                  Used to store the members of the next front
    for each p ∈ F_i
        for each q ∈ S_p
            n_q = n_q − 1
            if n_q = 0 then                q belongs to the next front
                q_rank = i + 1
                Q = Q ∪ {q}
i = i + 1
F_i = Q
End
```

**Fig. 6** Non-dominated sorting procedure

1. $L = |\delta|$
2. $for\ each\ individual\ i, set\ \delta[i]_{dist} = 0$
3. $for\ each\ objective\ m$
   a. $\delta = sort(\delta, m)\ in\ an\ ascending\ order$
   b. $\delta[1]_{dist} = \delta[L]_{dist} = \infty$
   c. $for\ i = 2\ to\ L - 1$
      $\delta[i]_{dist} = \delta[i]_{dist} + (\delta[i+1][m] - \delta[i-1][m])/(f_m^{max} - f_m^{min})$

**Fig. 7** Crowding distance calculation

### 3.1.1 Comparison Operators

For unconstrained multi-objective optimization, the NSGA-II employs the crowded-comparison operator ($\prec$). It directs the selection process at various stages of the algorithm. Every individual $i$ in the population has a nomination rank ($i_{rank}$) and a crowding distance rank ($i_{distance}$). It defines a crowded-comparison operator ($\prec$) as follows:

$$if\ (i_{rank} < j_{rank}) then\ i \prec j$$
$$or\ ((i_{rank} = j_{rank})\ and\ (i_{distance} > j_{distance}))$$

1. $Set\ t = 0\ and\ create\ a\ random\ parent\ population\ P^t\ with\ N$
2. $Perform\ binomial\ and\ mutation\ operator\ as\ in\ Fig.\ 5\ and\ Table\ II\ on\ P^t\ to\ obtain\ Q^t\ with\ size\ N$
3. $If\ the\ termination\ criteria\ is\ satisfied, stop\ and\ return\ P^t.$
4. $Set\ R^t = P^t \cup Q^t\ to\ combine\ two\ populations$
5. $Perform\ non-dominated\ sorting\ procedure\ for\ R^t\ to\ set\ the\ non-dominated\ fronts\ f_1, f_2, .., f_k.$
6. $For\ i = 1, .., k, repeat\ the\ following\ steps:$
   a. $Calculate\ crowding\ distance\ for\ each\ solution\ in\ f_i.$
   b. $Create\ P^{t+1}\ as\ follows:$
      i. $if\ |P^{t+1}| + |f_i| \leq N, then\ set\ P^{t+1} = P^{t+1} \cup f^t$
      ii. $if\ |P^{t+1}| + |f_i| > N, then\ add\ the\ least\ crowded\ N - |P^{t+1}|\ solutions\ from\ f_i\ to\ P^{t+1}.$
7. $Use\ binary\ tournament\ selection\ with\ constrained\ domination\ rule\ to\ select\ parents\ from\ P^{t+1}.$
8. $Apply\ binomial\ crossover\ and\ mutation\ operator\ to\ P^{t+1}\ and\ obtain\ new\ offspring\ population\ Q^{t+1}\ with\ size\ N$
9. $Set\ t = t + 1\ and\ go\ to\ Step\ 3.$

**Fig. 8** CGA and CGA_DE algorithm

It means that lower (better) rank is preferred between two individuals with different non-domination ranks. On the other hand, if both individuals have the same rank, it prefers individual with lesser crowded region.

Regarding the constrained multi-objective optimization, when comparing two individuals, the situation is somewhat different. Three cases can be observed: (1) one is feasible, the other is not; (2) both are infeasible; and (3) both solutions are feasible. For the constrained multi-objective optimization, NSGA-II modifies the definition of *domination* between two solutions as follows:

An individual $i$ is considered to constrained-dominate an individual $j$ under the following conditions:

(1) Individual $i$ is feasible and individual $j$ is infeasible.
(2) Both individuals are infeasible, but individual $i$ has a smaller constraint violation.
(3) Both individuals, $i$ and $j$ are feasible and individual $i$ dominates individual $j$ with crowded-comparison rule as follows:

$$if\ (i_{rank} < j_{rank})\ then\ i \prec j$$
$$or\ ((i_{rank} = j_{rank})\ and\ (i_{distance} > j_{distance}))$$

Now we are ready to outline the CGA and CGA_DE algorithms in Fig. 8.

As mentioned before, the difference between two algorithms comes from the different mutation operators used in step 2 and step 8 in Fig. 8. In the CGA, we immigrate some random new individuals with an amount of $MR_i \times \left|P^t\right|$ into the parent population. If the mutation rate $MR_i$ is equal to 0.02 with the population size $\left|P^t\right| = 100$, two individuals chosen from the population. Then, these individuals are regenerated randomly and uniformly within the boundaries of each dimension. In

case of CGA_DE algorithm, we employ the traditional mutation operator (rand/1/bin) of DE algorithms. In other words, after generating the offspring population, a certain percent of the child population randomly goes under a mutation operator. Again, if the mutation rate, $MR_i$ is equal to 0.02 with the population size $\left|P^t\right| = 100$, two individuals will be immigrated into the parent population. For each one, we randomly select three individuals from the parent population. The difference of two individuals is taken, multiplied by a uniform random number. $r$ between 0 and 1 in order to add to the third individual as in Eq. (28).

## 4 Computational Results

In this study, we proposed CGA and CGA_DE algorithms to deal with a multi-objective problem of self-sufficient floating neighborhood design. These algorithms are run for five independent replications on a computer, which has 2.6 GHz Intel core i7-6700HQ processor, 8 GBx2 DDR3 memory and 256 GB SSD. The population size is taken as $\left|P^t\right| = 100$. As a termination criterion, each algorithm was run for 100 generations. Initial values of the parameters are assigned to $CR_i = 0.9$ and $MR_i = 0.5$. With the self-adaptive procedure, we update these values with Eqs. (29) and (30) at each generation.

In Fig. 9, the standard deviations of five replications of the non-dominated solutions for each algorithm are shown. The standard deviation refers to the number of non-dominated solutions for each of 25 generations until 100th generation. From this graph below, we observe that there is no significant change after 75th generation until 100th generation. We observe zero standard deviation in this generation range for both algorithms. Thus, we determine the termination criteria as 100th generation for both algorithms.

Regarding the problem objectives, we analyzed minimum, average and maximum values of each objective function for each algorithm as can be found in Fig. 10, 11, 12, 13, 14, and 15. As it is seen from those graphs, each objective function has different characteristics in terms of their minimum, maximum, and average values generated by each algorithm.

Behaviors of cost-effectiveness are presented in Figs. 10 and 11. Until 50th generation, CGA and CGA_DE algorithms have presented different behavior in terms of average values. However, after 50th generation, both algorithms have generated similar results for the cost-effectiveness objective function.

Related to scenery objective function's results presented by both algorithms, maximum and average values are very similar as shown in Figs. 12 and 13. However, CGA_DE investigates a larger range of solutions, i.e., the results that are more diverse.

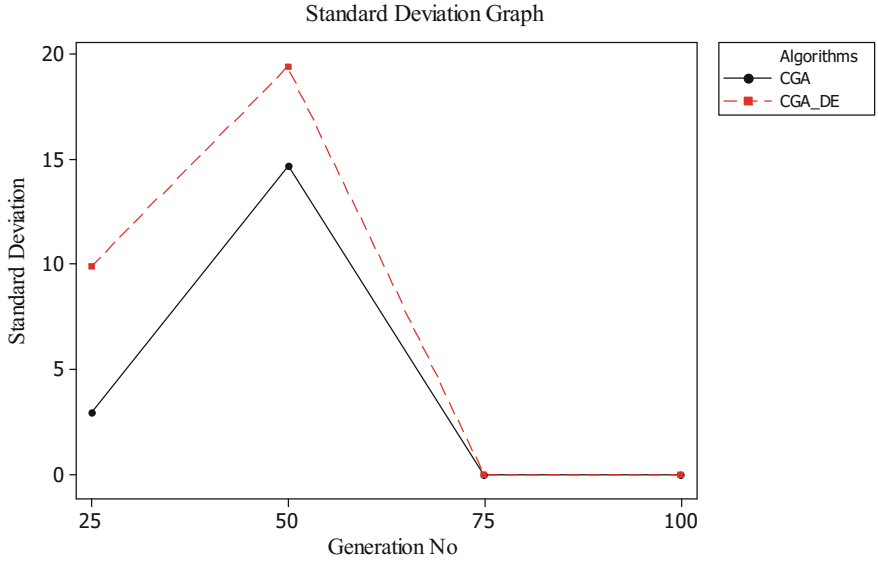


**Fig. 9** Standard deviation graph for CGA and CGA_DE

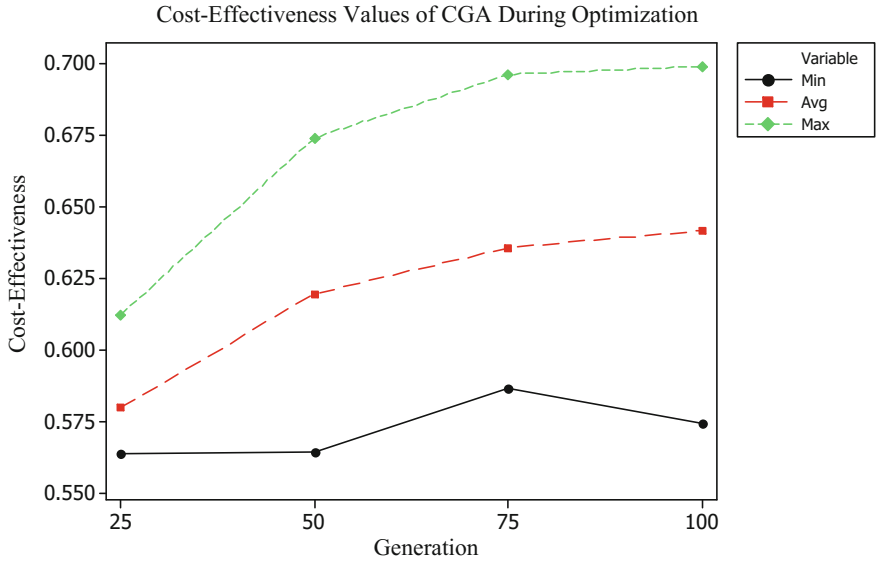


**Fig. 10** Cost-effectiveness values of CGA during the 100th generation

In terms of walkability objective, CGA_DE again presents larger range of results as shown in Fig. 14 and 15. CGA is able to achieve 0.25% in maximum value at 100th generation. In addition, the maximum values presented by CGA were almost

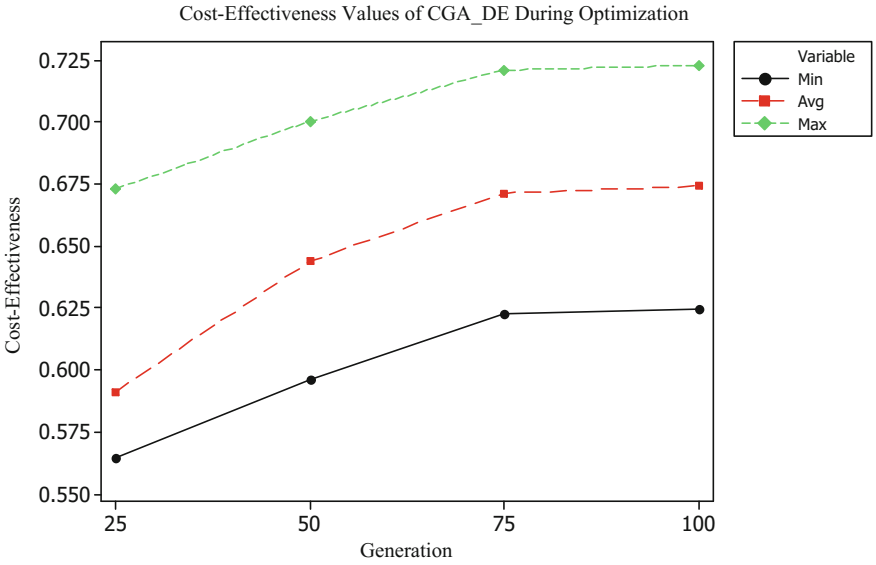


**Fig. 11** Cost-effectiveness values of CGA_DE during the 100th generation

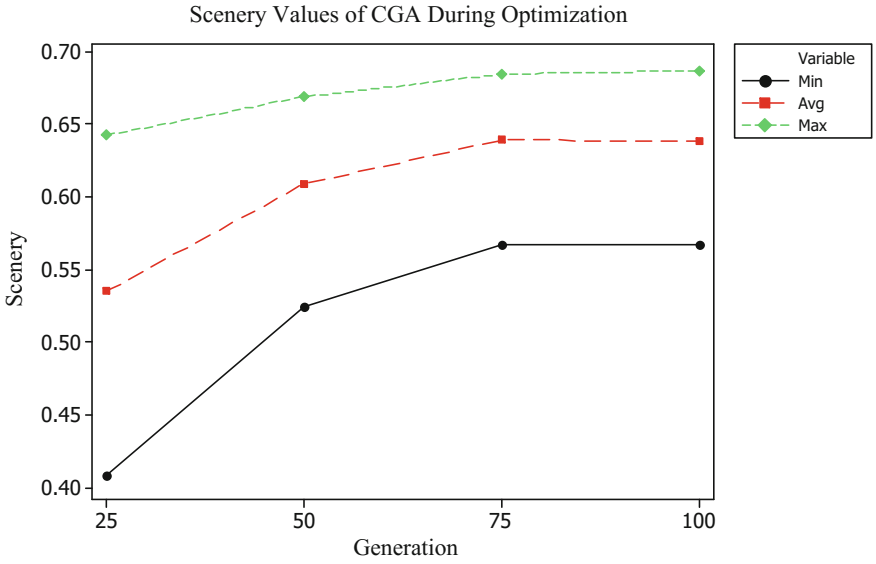


**Fig. 12** Scenery values of CGA during the 100th generation

the same in each generation. On the other hand, CGA_DE reaches 0.40% maximum value and presents changing results, which is an advantage for the design optimization problems.

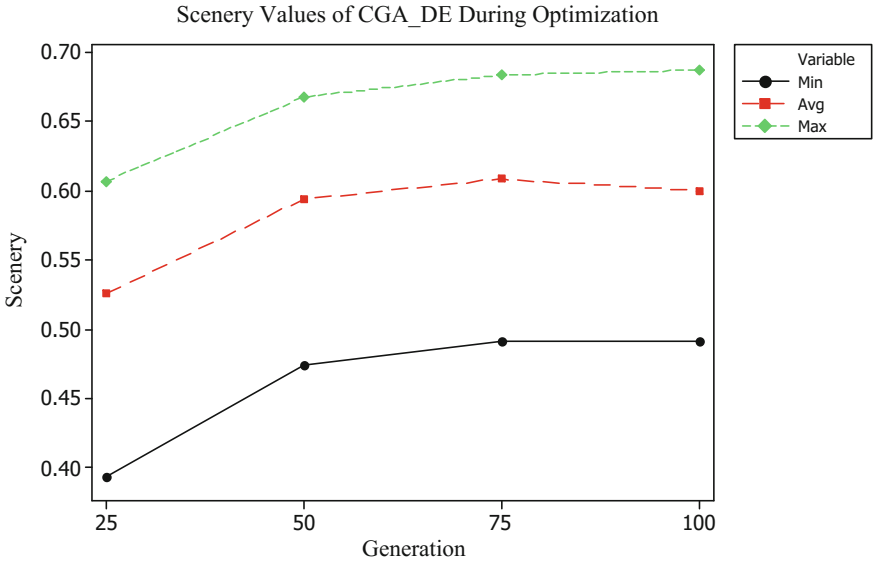


**Fig. 13** Scenery values of CGA_DE during the 100th generation

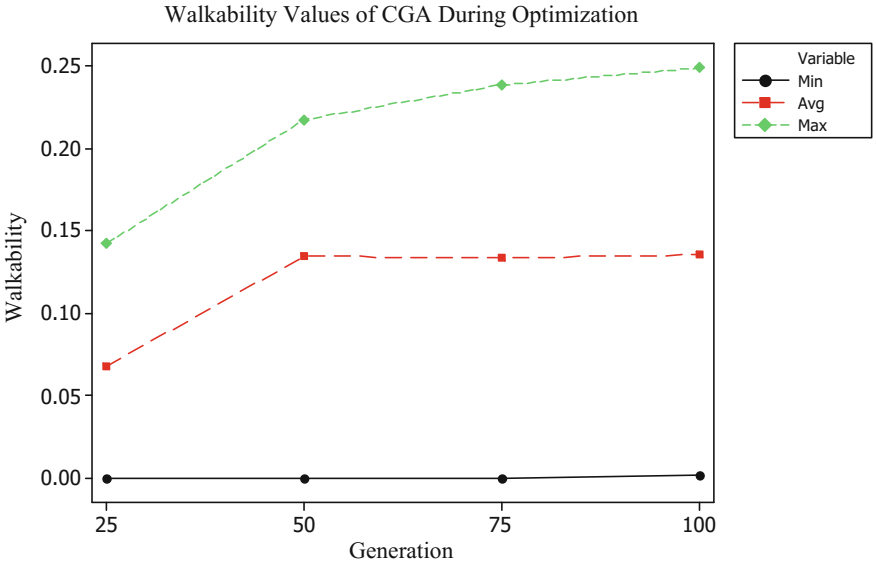


**Fig. 14** Walkability values of CGA during the 100th generation

Finally, the non-dominated solutions of CGA and CGA_DE after the 100th generation are given in Fig. 16. In the 3D scatter plot below, red dots correspond to the non-dominated individuals gathered from the CGA_DE; blue dots refer to the

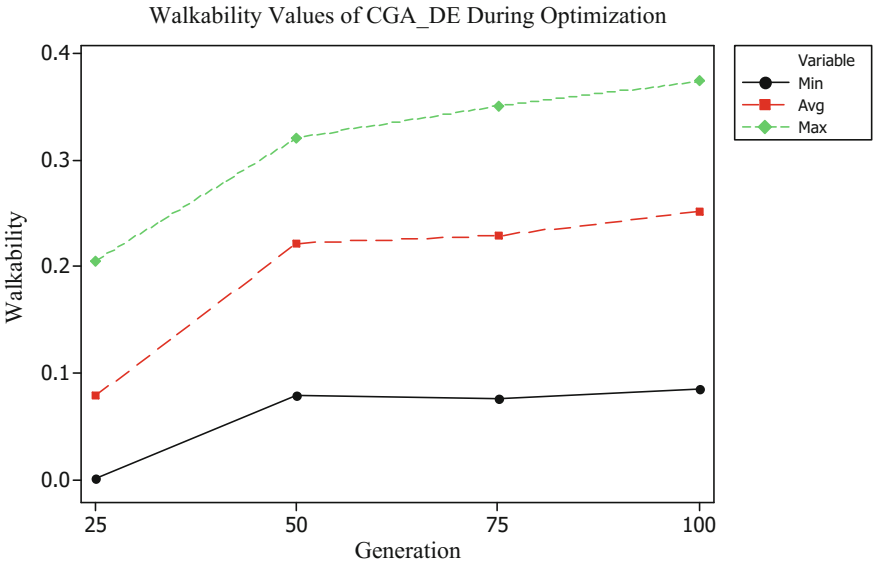


**Fig. 15** Walkability values of CGA_DE during the 100th generation

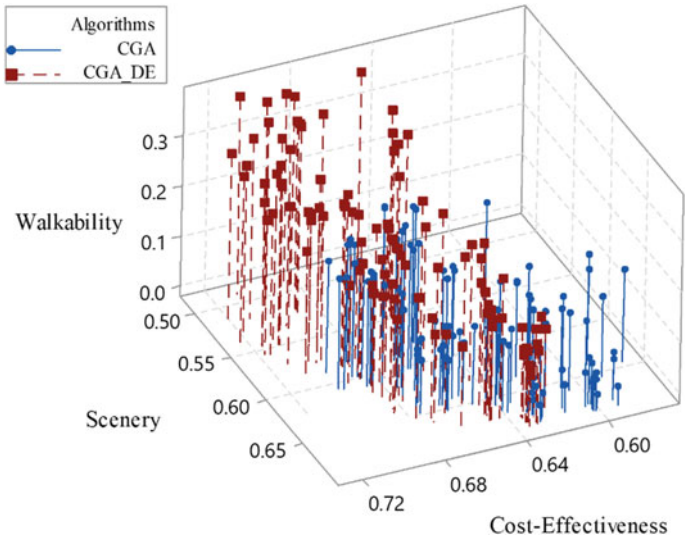


**Fig. 16** Non-dominated solutions in the 100th generation for CGA and CGA_DE

results obtained by CGA algorithm. The Pareto front approximation in both cases is investigated for all three objectives, supporting the claim of the objectives being conflicting with each other.

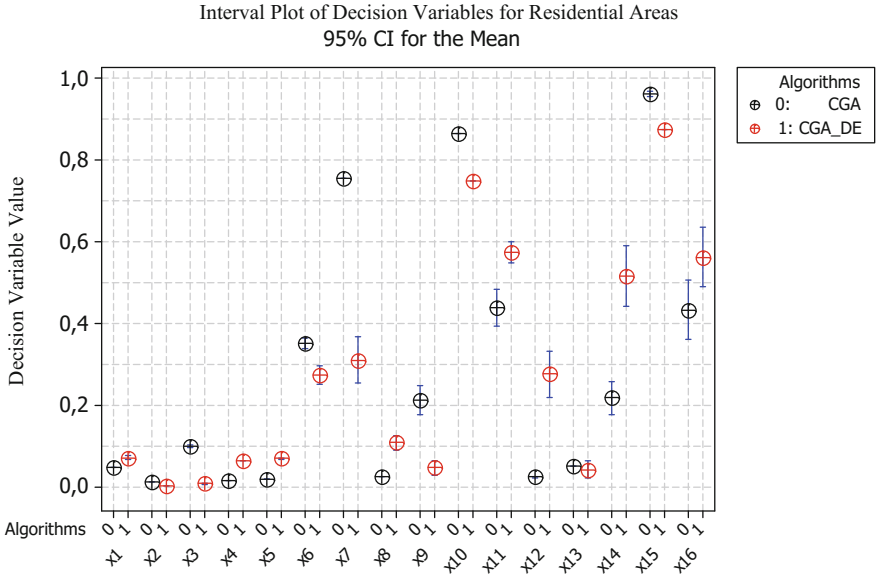


**Fig. 17** Interval plot of decision variables for residential areas

From the scatter plot above, it can be seen that solutions present very similar performances but different compositions of 64 design variables. With regard to the architectural qualities of the solutions, we need to analyze the influence of those 64 decision variables on each objective function for each algorithm. In this respect, we generated interval plots for each decision variable category considered in the optimization problem. By this way, we present behavior of each design variable for both CGA and CGA_DE algorithms. Thus, architectural specifications can be better discussed. All interval plots for different decision variable combinations are given in Fig. 17, 18, 19, and 20. In these figures, x1–x16 represents the densities of the residential areas in each cluster from one to sixteen. The percentages of the agricultural areas are represented by y1–y16. We used the notations of z1–z16 for public areas and t1–t16 for the percentages related to green areas.

As can be seen in Fig. 17, the percentages related to residential areas are almost the same in the clusters that are close to the coastline. In addition, the corresponding decision variables x1, x2, x3, x4, and x5 present lower results for CGA_DE algorithm than CGA algorithm. The percentages of residential areas in the clusters that are close to the offshore, in contrast, are higher in both algorithms. In few cases of decision variables with regard to the offshore clusters, the percentages are not as high as expected. This situation can be explained by the limited cluster capacities or the limitations caused by walkability objective.

The decision variables related to the agricultural areas present the highest percentages compared to the densities of other neighborhood functions. This can be explained by the high impact of the agricultural areas on both scenery objective and

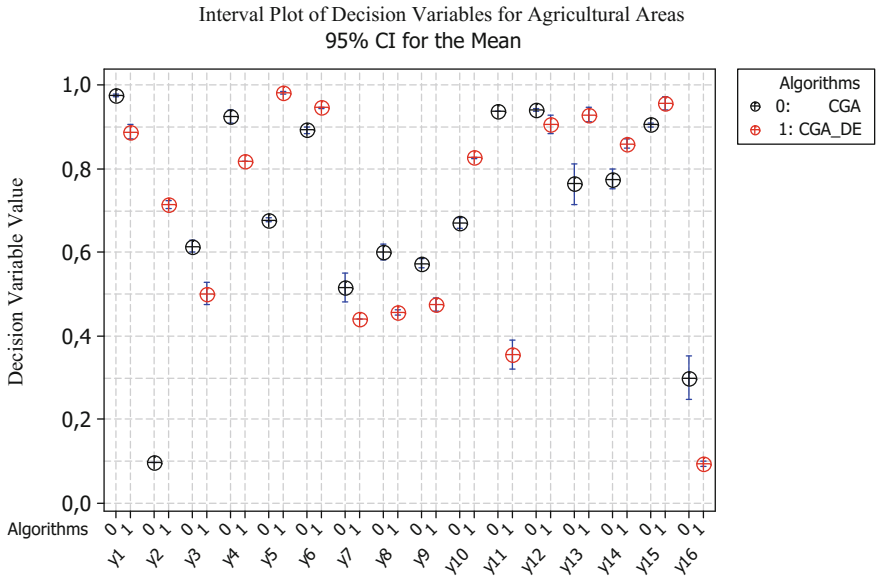


**Fig. 18** Interval plot of decision variables for agricultural areas

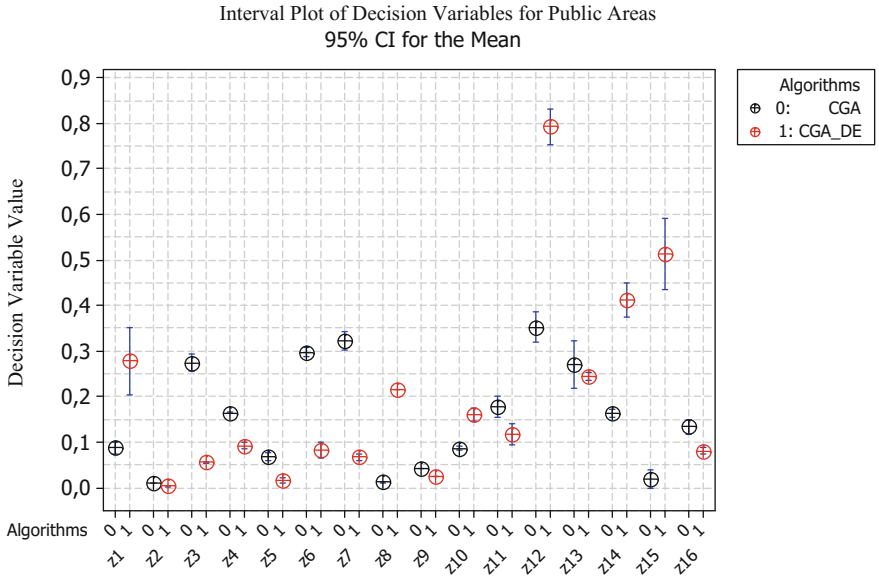


**Fig. 19** Interval plot of decision variables for public areas

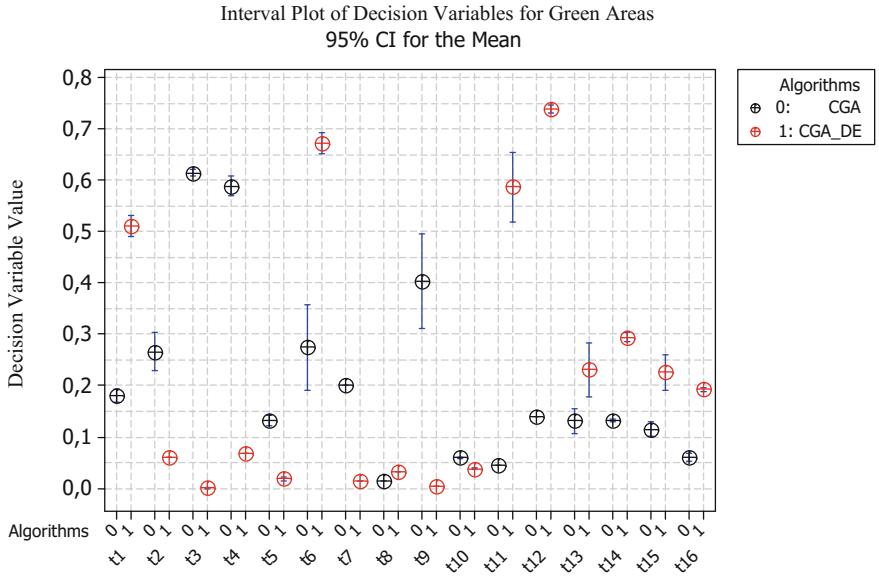


**Fig. 20** Interval plot of decision variables for green areas

problem constraints. From the point of algorithm comparison, CGA_DE presents more uniform results than CGA. As can be seen in Fig. 18, CGA picks the percentages of the agricultural areas in some cases.

The percentages of public areas consist just in walkability objective as controllable variables. In contrast to the agricultural areas, public areas have the lowest percentages in total compared to the percentages of other areas as shown in Fig. 19.

The occupancy rates of green areas in clusters that are close to the coastline are higher than in the cluster that are close to the offshore. It means that the effect of green areas on scenery is relatively high. For the decision variables related to green areas, CGA presents more changing results than CGA_DE as shown in Fig. 20.

## 5 Generative Model and Form-Finding

Generative model of the floating neighborhood problem has been created in the Grasshopper algorithmic modeling environment [14]. Grasshopper is a plug-in for Rhinoceros, which is a well-known CAD program. This section consists of two parts, which are parametric definition of functions' distributions based on rules and form generation of the chosen result.

**Fig. 21** Representation of info graphics for each cluster

## *5.1 Parametric Definition*

In the parametric definition of floating neighborhoods, the percentages of each neighborhood function in each cluster are generated based on rule-based decision-making. In order to represent densities, we use info graphics. The info graphics are illustrated in the Rhino model with four different colors in Fig. 21. This figure belongs to one design alternative gathered after optimization, which presents both walkable and scenic characteristic. In so-called infographics, pink, purple, yellow, and green colors correspond to residential areas, public areas, agricultural areas, and green areas, respectively.

## *5.2 Form Generation*

In this section, we explained how the form of floating neighborhood is generated in relation with infographics. Voronoi diagram algorithm is used for further elaborations of the floating neighborhood. As can be seen in Fig. 22, each cluster has different sizes based on their different capacities.

**Fig. 22** 16 clusters with different sizes [7]

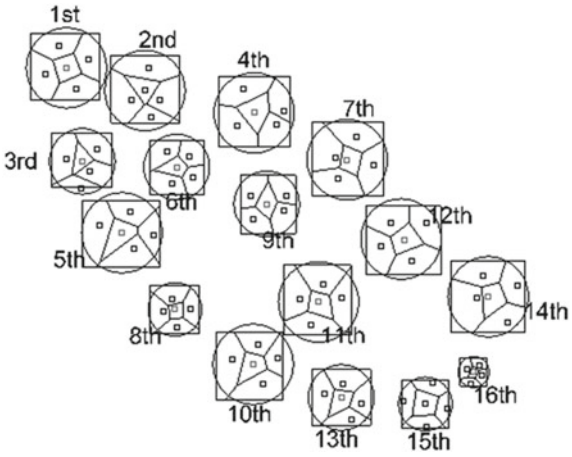


**Fig. 23** Voronoi diagram for each cluster [7]

After determination of clusters' sizes, Voronoi diagram algorithm [15] divides each cluster into parts that each one corresponds to each neighborhood function. Figure 23 shows the Voronoi diagrams within the floating neighborhood.

After the final adjustments on the generative model, computer rendering are get prepared. As can be seen in Figs. 24, 25 and 26, final self-sufficient floating neighborhood design is indeed plausible.

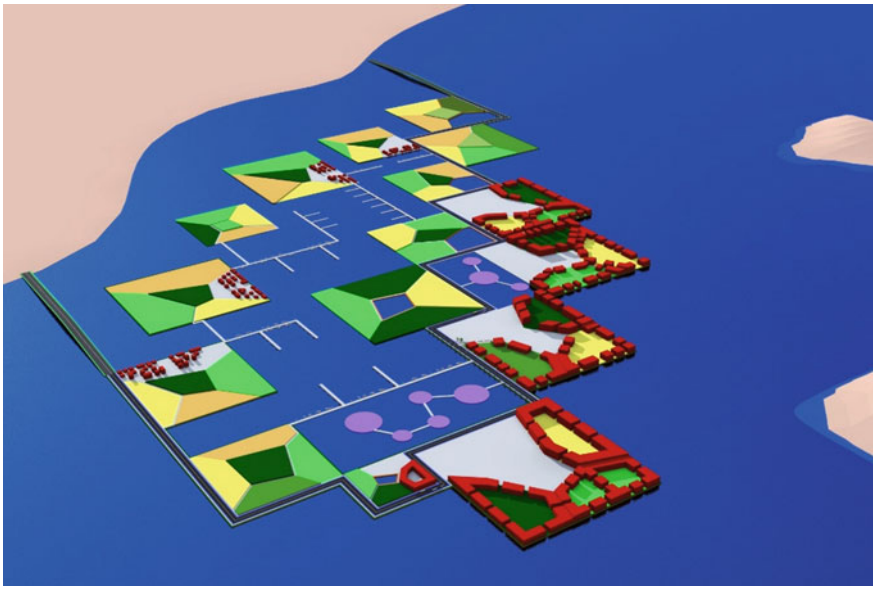

**Fig. 24** Computer rendering from general view to floating neighborhoods [7]

**Fig. 25** Computer rendering from agricultural areas

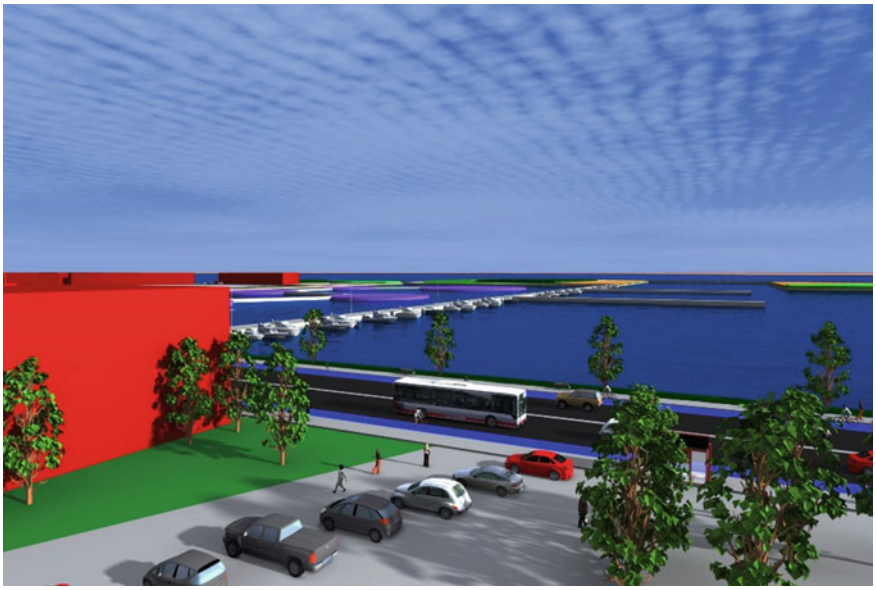

**Fig. 26** Computer rendering from residential and public areas

## 6 Conclusion

In this study, two different evolutionary algorithms, namely CGA and CGA_DE are implemented to solve the complex design problem of floating neighborhoods. The design goals consist of both architectural and engineering aspects, which are cost-efficiency, scenery and walkability. These objectives are conflicting. Thus, the design methodology combines the floating structures design practices with computational intelligence and utilizes computational design strategies. The comparison of algorithms is performed through graphical representations. Both algorithms have presented competitive results. However, CGA_DE found more spread-out solutions than CGA for this multi-objective constrained real-parameter floating neighborhood design problem. In addition to objective function analysis, we also discussed the results of decision variables to analyze architectural qualities of the solutions. As a result, we achieved Pareto front approximations for both algorithms with non-dominated individuals, and both feasible and indeed plausible architectural design solutions.

## References


1. Pachauri, R. K., Meyer, L., & Intergovernmental Panel on Climate Change (Eds.). (2015). *Climate change 2014: Synthesis report*. Geneva, Switzerland: Intergovernmental Panel on Climate Change.
2. Field, C. B., & Intergovernmental Panel on Climate Change (Eds.). (2012). *Managing the risks of extreme events and disasters to advance climate change adaption: Special report of the Intergovernmental Panel on Climate Change*. New York, NY: Cambridge University Press.
3. DeltaSync and The Seasteading Report: Design input, location design. Retrieved December 3, 2017, from https://www.seasteading.org/floating-city-project/.
4. Watanabe, E., Wang, M., Utsunomiya, T., & Moan, T. (2017). Very Large Floating Structures: Application, Analysis and Design; Technical Report No. 2004-02. Retrieved December 3, 2017, from http://www.eng.nus.edu.sg/core/Report%20200402.pdf.
5. Floating City concept by AT Design Office features underwater roads and submarines, Dezeen Magazine. Retrieved December 3, 2017, from https://www.dezeen.com/2014/05/13/floating-city-at-design-office/.
6. Floating settlements proposal by Baca Architects. Retrieved December 3, 2017, from https://www.dezeen.com/2017/06/09/video-baca-architects-floating-architecture-homes-movie/.
7. Kirimtat, A., Chatzikonstantinou, I., Sariyildiz, S., & Tartar, A. (2015). Designing self-sufficient floating neighborhoods using computational decision support. In *CEC 2015*, Sendai, Japan.
8. Walk Score Website. https://www.walkscore.com/.
9. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6,* 182–197.
10. Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. In *Eurogen, 2001* (pp. 95–100).
11. Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.
12. Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Žumer, V. (2006). Selfadapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation, 10*(6), 646–657.
13. Brest, J. (2009). Constrained real-parameter optimization with e-self-adaptive differential evolution. In E. Mezura-Montes (Ed.), *Constraint-handling in evolutionary optimization*. *Studies in computational intelligence series* (Vol. 198). Springer.
14. Grasshopper, Algorithmic Modeling for Rhino. http://www.grasshopper3d.com/.
15. Brandt, J. W., & Algazi, V. R. (1992). Continuous skeleton computation by Voronoi diagram. *CVGIP: Image understanding, 55*(3), 329–338.