Annals of Operations Research 123, 203–222, 2003
© 2003 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Genetic Algorithm Based Approach for the Uncapacitated Continuous Location–Allocation Problem *

S. SALHI s.salhi@bham.ac.uk
The University of Birmingham, UK

M.D.H. GAMAL mdhgamal@unri.ac.id
The University of Riau, Pekanbaru, Indonesia

Abstract. A GA-based approach is introduced to address the continuous location–allocation problem. Selection and removal procedures based on groups of chromosomes instead of individual chromosomes are put forward and specific crossover and mutation operators that rely on the impact of the genes are proposed. A new operator that injects once in a while new chromosomes into the population is also introduced. This provides diversity within the search and attempts to avoid early convergence. This approach is tested on existing data sets using several runs to evaluate the robustness of the proposed GA approach.

Keywords: GA heuristic, location, continuous space

1. Introduction

In this paper, we propose a new implementation of GA that includes strategies for the selection and the removal of parents, and schemes for mating and reproducing while maintaining diversity within the population. The continuous location–allocation problem is used as a platform to test these ideas within a GA framework. We first briefly describe this continuous location problem and then present our approach to solve this problem.

In this location–allocation problem, we are interested in finding the location of p facilities in continuous space in order to serve customers at n fixed points as well as the allocation of each customer to the facilities so that total transportation costs are minimized. We assume there is no restriction on the capacity of the facilities. The location–allocation model is formulated as follows:

$$\text{Minimize} \sum_{i=1}^{p} \sum_{j=1}^{n} w_{ij} d(\boldsymbol{X}_i, \boldsymbol{a}_j) \tag{1}$$

* An earlier version of this paper was given at *OR43*, Bath.

$$\text{subject to } \sum_{i=1}^{p} w_{ij} = q_j, \quad j = 1, \ldots, n,$$
$$w_{ij} \geqslant 0, \qquad \text{for all } i \text{ and } j,$$

where w_{ij} is the quantity assigned from facility i to fixed point j, $d(\boldsymbol{X}_i, \boldsymbol{a}_j)$ is the Euclidean distance from the location (coordinates) of facility i, (X_i, Y_i), to the location of a customer at fixed point j, (x_j, y_j), and q_j is the demand or weight of customer j.

The problem is also known as the multisource Weber problem. Under the assumption that there are no capacity constraints at the new facilities, it can be shown that the demand at each point is satisfied by the nearest facility at minimum cost. Cooper (1963) proves that the objective function (1) is neither concave nor convex, and may contain several local minima. Hence, the multisource Weber problem falls in the realm of global optimisation problems. Heuristic methods have been shown to be the best way to tackle larger NP-hard problems. Modern heuristics such as Simulated Annealing, Tabu Search, Genetic Algorithms (GA), Variable Neighborhood Search, and Ant Systems increase the chance of avoiding local optimality. In this study we opt for a population based method (GA) as it has the flexibility of producing more than one solution which is preferred in solving strategic type problems like the location problem. It may be argued that multi-start type heuristics or node exchange heuristics also generate alternative solutions. However, such solutions are found as by product of the search and hence do not represent the solution space as GA does except when a large number of effective diversification schemes are incorporated into those methods.

The paper is organised as follows: in the next section we briefly review the literature and in section 3 we present our GA-based approach where we emphasize the main schemes within the method. In section 4 we provide our computational results and in the last section we summarise our findings and highlight some issues that may need to be addressed.

2. Previous research

Many heuristic methods have been proposed in the literature for the solution of the multisource Weber problem. The first heuristic is the well-known iterative location–allocation algorithm of Cooper (1964). Cooper's heuristic generates p subsets of fixed points and then solves each one using the exact method for solving a single-facility location problem. We shall briefly discuss Cooper's algorithm since this method is used as a basis for cost evaluation of our GA heuristic.

The fixed points set is divided into p subsets. For each of these p subsets, using an initial facility location, the exact location method is applied to find the optimal single facility location. Each fixed point is then reallocated to the nearest facility. After all fixed points have been completely reallocated, the exact location method is applied again to improve the location of those facilities where the set of customers assigned has changed. This process, alternating between the location and the allocation phases, is repeated until no further improvement can be made.

The solution found by the alternate algorithm is a local minimum. Eilon, Watson-Gandy and Christofides (1971) showed that for a problem with $p = 5$ and $n = 50$, using 200 randomly generated starting solutions, 61 local optima were found, and the worst solution deviates from the best one by 40.9%. In order to have the closest local minimum to the optimal one, the method is repeated several times using different starting locations at random. The repetition of Cooper's alternate algorithm several times is also known as the *multi-start alternate algorithm*.

Kuenne and Soland (1972) created a branch-and-bound algorithm which produces an exact solution for problems with 25 fixed points and 1–5 facilities. Love and Morris (1975) develop the set reduction method and a p-median algorithm to solve the multi-source Weber problem with rectangular distance. Their method gives the exact solution to problems with 35 fixed points and 2 facilities. Drezner (1984) solved the planar two-median and two-center problems optimally. He showed that in these two location problems any two customers sets can be separated by a line. Since the optimal location of a facility within each set can be found optimally, the problem reduces to finding an efficient way of defining these lines. Drezner proposed two algorithms, one for each problem, to determine efficiently these lines. Computational results for problems up to 100 customers were found efficiently. Rosing (1992) proposes a method to solve the (generalised) multisource Weber problem. He divides the set of fixed points into non-overlapping convex hulls and generates the list of all feasible convex hulls where each fixed point must belong to exactly one of those convex hulls. The cost function associated with each convex hull is computed as a single Weber problem. This method produces the optimal solution to problems with up to 30 fixed points and 6 facilities. Given the restrictive use of these optimal methods, heuristics are used. Brimberg and Mladenović (1996b) adopt a tabu search approach to the problem. Hansen, Mladenović and Taillard (1998) solve the continuous location–allocation problem via the p-median problem by considering all fixed points as potential facility sites. A variable neighborhood search algorithm, is designed by Brimberg and Mladenović (1996a). The idea is to adjust the size of the neighborhood around the current solution systematically. A specified number of points is randomly generated from the neighborhood. If one of these points gives a descent move, it is selected; otherwise the neighborhood is enlarged in order to enhance the search. This method uses Cooper's alternate algorithm to carry out the local descent. Another version of this method which allows an ascent move is also given in their paper. Enhancements of some proposed heuristics (Bongartz, Calamai and Conn, 1994; Brimberg and Mladenović, 1996b; Hansen, Mladenović and Taillard, 1998; Houck, Joines and Kay, 1996; Brimberg and Mladenović, 1996a) are put forward by Brimberg et al. (2000). Gamal and Salhi (2001) proposed a constructive heuristic that attempts to avoid local optimality by first locating far apart facilities and then adaptively forbidding/freeing certain regions by shrinking/extending the size of these regions using previous knowledge. Recently Gamal and Salhi (2002) presented a learning scheme which uses previous solutions to discretize the continuous space into well-defined rectangular shape cells. This cells-based technique considers frequency of occurrence of

already found configurations as well as a measure of compatibility between the facilities.

There is a shortage of papers on the use of GA for continuous location–allocation problems. The first attempt is made by Houck, Joines and Kay (1996). In their implementation a floating point (real number) representation is used where p pairs of the real numbers (x, y) represent the locations of p facilities to be located. Lower and upper bounds of x and y values are used to direct the search to possible values of x and y, respectively. The initial population is created by generating N sets of p real random numbers. Several crossover and mutation techniques are applied to produce the offspring. For larger problems, it is statistically shown that their results, when compared to the ones obtained by random restart and the H4 heuristic of Love and Juel (1982), are better and use less computational effort. Brimberg et al. (2000) adopt the GA developed by Houck, Joines and Kay (1996), except that when doing crossover, they consider the sparsity of the facility locations and avoid duplication of good locations. Using their *parent* selection criteria, which suits the survival-of-the-fittest strategy, they pick two solutions from the population as *parent* 1 and *parent* 2. For $i = 1, \ldots, p$, the child solution produced contains the ith facility location of *parent* 1 or *parent* 2, depending on which one satisfies the minimum separation distance. They consider the smallest distance between two customers as the minimum separation distance. Cooper's alternate algorithm is used as their mutation operator to obtain the local minimum. In both implementations, (Houck, Joines and Kay, 1996; Brimberg et al., 2000), the fitness value of a given chromosome is defined as the objective function value of such a chromosome obtained after applying Cooper's alternate algorithm. The main steps of a typical GA implementation are given in figure 1.

In this study the main steps of our GA heuristic are similar to the ones outlined in figure 1, except that we exploit each step further by introducing new schemes. We investigate the construction of the initial population, the parent selection process, the implementation of the genetic operators, and the way diversity and early convergence are controlled.

Step 1. Create the initial population by generating K sets of p random facility locations.

Step 2. Evaluate the fitness of each chromosome in the population by applying Cooper's alternate algorithm to the chromosome.

Step 3.

Repeat

(i) Apply parent selection procedures.

(ii) Apply genetic operators to produce new chromosomes.

(iii) Add these new chromosomes into the population.

(iv) Apply survival (removal) procedures to reduce (maintain) the population size.

Until stopping criteria are satisfied.

Figure 1. The basic skeleton of a GA.

3. A GA-based heuristic

3.1. Representation

In traditional GA, a binary coding is used to represent a chromosome. In practice, it is not always appropriate to convert a solution to a binary representation, besides having to maintain feasibility when applying the operators such as mutation and crossover. Repair mechanisms and penalty functions are usually introduced to deal with such a problem. For instance, experiments show that integer representation is more suitable for the TSP, see (Michalewicz, 1992).

In continuous location problems, a binary representation may result in locating two facilities which are very close to each other. As in (Houck, Joines and Kay, 1996; Brimberg et al., 2000) we also use a real number representation where a chromosome consists of p (x, y) pairs representing the sites of facilities to be located, and p is the number of facilities. For instance this is represented as follows:

$$\text{Chromosome:} \quad \big[(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_i, y_i), \ldots, (x_p, y_p)\big],$$

where the coordinate (x_i, y_i) denotes the location of the ith facility, $i = 1, \ldots, p$.

The fitness value used is the cost of allocating customers to those open facilities. This is then improved using the alternate location–allocation procedure of Cooper starting from the current locations as represented by our chromosomes.

3.2. Initial population

The initial population is usually constructed by choosing the real numbers randomly between the lower and upper bound of the abscissa and the ordinate of the customer locations. The initial population can be constructed as follows:

$$\begin{array}{lllllll}
\big[(x_{11}, y_{11}), & (x_{12}, y_{12}), & (x_{13}, y_{13}), & \ldots, & (x_{1i}, y_{1i}), & \ldots, & (x_{1p}, y_{1p})\big] \rightsquigarrow f_1, \\
\big[(x_{21}, y_{21}), & (x_{22}, y_{22}), & (x_{23}, y_{23}), & \ldots, & (x_{2i}, y_{2i}), & \ldots, & (x_{2p}, y_{2p})\big] \rightsquigarrow f_2, \\
 & & & \vdots & & & \\
\big[(x_{k1}, y_{k1}), & (x_{k2}, y_{k2}), & (x_{k3}, y_{k3}), & \ldots, & (x_{ki}, y_{ki}), & \ldots, & (x_{kp}, y_{kp})\big] \rightsquigarrow f_k, \\
 & & & \vdots & & & \\
\big[(x_{K1}, y_{K1}), & (x_{K2}, y_{K2}), & (x_{K3}, y_{K3}), & \ldots, & (x_{Ki}, y_{Ki}), & \ldots, & (x_{Kp}, y_{Kp})\big] \rightsquigarrow f_K,
\end{array}$$

where (x_{ki}, y_{ki}) are the (x, y) coordinates of the ith selected facility, $i = 1, \ldots, p$, of the kth chromosome, $x_{ki} \in [x_{\min}, x_{\max}]$ and $y_{ki} \in [y_{\min}, y_{\max}]$, f_k is the fitness function of the kth chromosome, and K is the maximum number of chromosomes in the population (also known as the population size).

It is worth noting that if customer locations are spread uniformly in the region, choosing random points in the region may not be a bad strategy. However, if customers are clustered, as may happen in practice, choosing the points randomly without knowledge of the clusters may result in poor initial locations. For instance the facilites can be

situated in large empty regions or far away from large customers. Figure 2 illustrates this situation.

As we do not know beforehand the spread of the fixed points, to handle the case of clusters, a scheme that identifies such possible spread for selecting the initial points is proposed. Specifically, we first approximate the cumulative probability distribution of the number of customers in the cells and then we randomly choose these fixed points based on that probability distribution rather than the uniform distribution. One way of dealing with this issue is to cover the entire region with $k_0 \times k_0$ rectangular cells where the width of the cell in the x-axis is $W_x = (x_{\max} - x_{\min})/k_0$ and the length is $W_y = (y_{\max} - y_{\min})/k_0$. In this study we consider the number of divisions, for the x-axis and the y-axis to be constant, say k_0. A cell is defined by its bottom-left corner. Let the coordinates of the bottom-left corner of the jth cell be (X_j, Y_j), $j = 1, \ldots, k_0^2$. We record the cells as follows: cell 1 has its bottom-left corner $(X_1, Y_1) = (x_{\min}, y_{\min})$ and subsequent cells, say cell j as $(X_j, Y_j) = (x_{\min} + k_x W_x, y_{\min} + k_y W_y)$ where $j = k_y \text{mod}(k_0) + k_x$.

We choose the value of k_0 based on the argument that if p cells are optimally found then $k_0 = \sqrt{p}$ and hence there will be one facility in each cell. But as this claim is hard to realise we provide flexibility by splitting each cell into four smaller cells yielding $k_0 = 2\sqrt{p}$. For smaller values of p we bounded this value by 10 to guarantee at least 100 cells. This idea is usually adopted in the quadtree techniques in computer science and in GIS, see (Hodgson and Salhi, 2003).

$$k_0 = \begin{cases} 10 & \text{if } p \leqslant 25, \\ 2\sqrt{p} & \text{otherwise.} \end{cases}$$

We record the number of points in each cell, n_j, $j = 1, \ldots, k_0^2$. We then construct the cumulative probability distribution of the cells, starting from cell 1 and finishing at cell k_0^2. The generation of the coordinates of the ith facility (x_i, y_i) is given below.

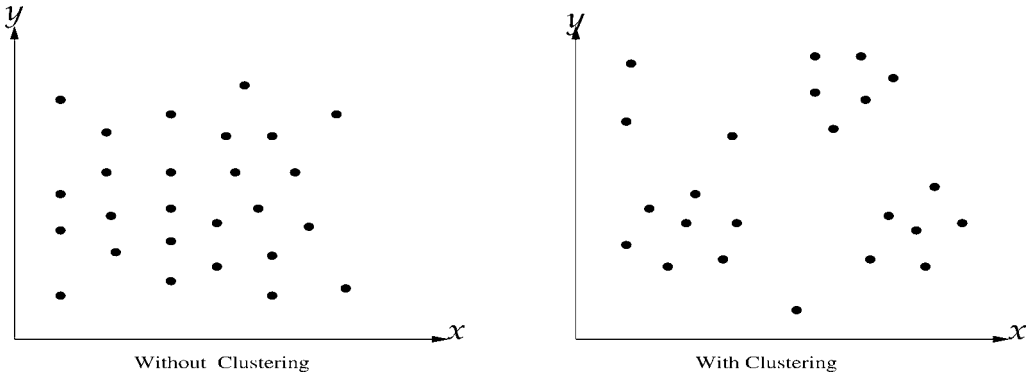

Figure 2. The spread of customer locations.

Generation of the initial locations

- Construct k_0^2 cells of width W_x and length W_y.
- Record the number of fixed points in each cell and construct its corresponding cumulative probability distribution.
- Choose a random number r in (0,1).
- Determine the corresponding cell j based on the cumulative probability.
- Generate x_i and y_i randomly in the ranges $[X_j, X_j + W_x]$ and $[Y_j, Y_j + W_y]$, respectively, using the uniform distribution.

The above generation scheme will produce, with high probability, location points where clusters exist. These locations can then be improved using Cooper's alternate location–allocation procedure. It is worth noting that the validity of such an operation can be affected by the size of the cells and the positions of the their boundaries. The main steps of the construction of the initial location are summarised in figure 3.

3.3. Parent selection

Parent selection can be critical to the success of GAs. Though the selection needs to follow the survival-of-the-fittest strategy, in some situations, it is difficult to distinguish between the solution quality of some of the chromosomes if their objective function values are very similar. One commonly used approach is to transform the objective function into a fitness function by stretching the differences via some appropriate analytical transformations or some ranking based indices, see (Michalewicz, 1992). In this study we overcome this difficulty by grouping the chromosomes of the entire population in classes ordered according to their fitness values. Within each class all chromosomes are treated the same and can be chosen randomly in a uniform way. However, each class is associated with its own proportion of acceptance, for instance the top class has a higher value than the bottom class. A roulette-wheel-like selection procedure is then applied to these classes instead of the individual chromosomes.

Let the objective function f_i of K solutions be nonincreasingly ordered as:

$$f_1 \leqslant f_2 \leqslant f_3 \leqslant \cdots \leqslant f_k \leqslant \cdots \leqslant f_K.$$

Our previous experiment (Gamal and Salhi, 2001) shows that the values of f_k are usually very close to each other. In this study, we group the entire population into 3 classes α_1, α_2, and α_3 consisting of good, not so good and bad chromosomes, respectively. This classification is adopted in such a way that the, first n_1 of f_k is represented by $\alpha_1(t)$,

Step 1.	Choose p pairs of (x, y) using the generation scheme given above.
Step 2.	Apply Cooper's alternate algorithm on these p locations found in step 1 to obtain the corresponding fitness value.
Step 3.	Repeat step 1 K times to make up the entire initial population.

Figure 3. The generation of the initial population.

Step 1.	Order the population nonincreasingly based on the chromosome fitness values measured by the value of the objective function of the solutions.
Step 2.	Construct 3 groups consisting of good, not so good and bad chromosomes of the population based on the proportions α_1, α_2, and α_3, respectively.
Step 3.	Select the group by applying the roulette wheel selection to these 3 groups.
Step 4.	Pick a solution uniformly randomly from the selected group.
Step 5.	Repeat step 4 until the number of required parents is reached.

Figure 4. Parent selection procedure.

the second n_2 by $\alpha_2(t)$, and the last n_3 by $\alpha_3(t)$ where t denotes the tth generation, and $n_1 > n_2 > n_3 = K - n_1 - n_2$:

$$\underbrace{f_1, f_2, f_3, \ldots}_{\alpha_1}, \underbrace{f_k, \ldots, f_l}_{\alpha_2}, \underbrace{\ldots, f_{K-1}, f_K}_{\alpha_3}.$$

For instance we initially use a 60%, 30% and 10% split and these values are kept for a certain number of iterations, say 40% of the maximum number of generations, after that they are adjusted, see the section on the computational results for details. This is performed by gradually increasing α_1 while decreasing α_2 and α_3. Probabilistically, the solutions in α_1 have more chance to be selected than the ones in the α_2 and α_3 groups subsequently. The main steps of the parent selection procedure are given in figure 4.

Figure 5 provides an overall representation of the generation of a new population. Some of the blocks in the figure will be described later in this section.

3.4. Removal procedure

Before inserting an offspring into the population we remove some chromosomes from the current population. In this study we aim to keep the population size constant. For the removal procedure we use a technique similar to the one used in parent selection which satisfies the survival-of-the-fittest strategy. In the removal procedure we group the objective function values in such a way that the first n_3 of f_k is represented by $\alpha_1(t)$, the second n_2 by $\alpha_2(t)$, and the last n_1 by $\alpha_3(t)$ where t denotes the tth generation, and $n_3 > n_2 > n_1 = P - n_3 - n_2$:

$$\underbrace{f_1, f_2, f_3, \ldots}_{\alpha_3}, \underbrace{f_k, \ldots, f_l}_{\alpha_2}, \underbrace{\ldots, f_{K-1}, f_K}_{\alpha_1}.$$

Values α_1, α_2, and α_3 are kept constant for a certain number iterations, and then adjusted. This is done in the same manner as in the parent selection procedure. Probabilistically, the solutions in α_1, which are the less fit chromosomes in the population, have a higher chance to be removed from the population. However, such low quality chromosomes are not automatically as some of these chromosomes may inherit important genes that can be useful for diversity reasons. The main steps of this procedure are as depicted in figure 4 and hence not repeated here.

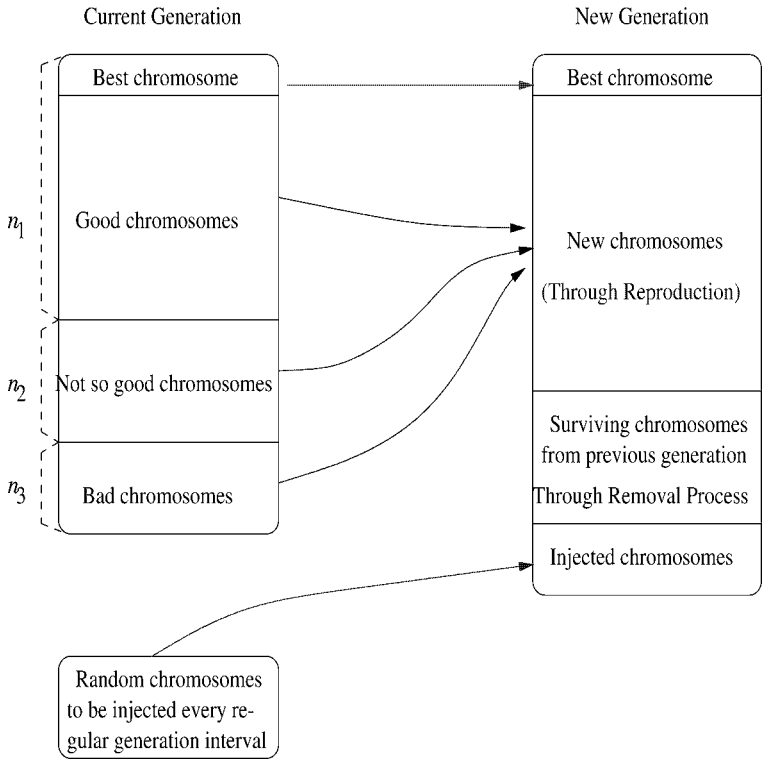

Figure 5. New generation composition.

3.5. Mutation

The mutation operator tends to make small random changes in one chromosome to form one offspring. For each chosen chromosome, we first determine those genes that will be mutated (phase 1) and then produce a mutation scheme that selects the parent for mutation (phase 2).

Phase 1. The selection of genes for mutation
Parent 1 chromosomes are those candidates whose genes (facilities) will be mutated. These chromosomes are selected randomly among those in the top group as these chromosomes yield already good quality fitness values and hence the mutation operator will, hopefully by affecting only one gene at a time, improve the solution further. Note that generally, in the usual GA procedures, every gene has an equal chance to be mutated. In our implementation, we guide the search by mutating genes which satisfy the criteria described below. Let a chromosome be represented by a vector:

$$\big((x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_p, y_p)\big),$$

where the coordinate (x_i, y_i) denotes the location of the ith facility. Let the vector

$$\left(\widehat{N}_1, \widehat{N}_2, \widehat{N}_3, \ldots, \widehat{N}_p\right)$$

denote a measure of how busy a given facility is. We define $\widehat{N}_i = \sqrt{N_i W_i} \quad \forall i = 1, \ldots, p$, where N_i and W_i denote the number of customers and the total weight served by the ith facility, respectively. In the mutation process, we introduce this measure to guide the search in selecting those facilities that will be mutated. This measure will differentiate between two facilities which are very close to each other, and then the mutation procedure will mutate the one which has a very small or a very large value of this measure (in other words, the one that serves few customers or many customers). The potential genes for mutation are found using the following procedure which is given below.

(a) Choose a chromosome k randomly from the top group α_1.

(b) Order the genes (facilities) of chromosome k nonincreasingly with respect to the $\widehat{N}_i$ values, $i = 1, \ldots, p$.

(c) Construct E_k, the set of genes to be mutated, as $E_k = \{i = 1, \ldots, p$ such that $\widehat{N}_i \leqslant \overline{N} - \gamma\sigma$, or $\widehat{N}_i \geqslant \overline{N} + \gamma\sigma\}$, where $\overline{N}$ and σ denote the mean and the standard deviation, and γ is a correction factor emphasizing the effect of the width of the range.

According to (Lindley and Scott, 1989), $\gamma = 1$ provides about 68% acceptance within the range, whereas $\gamma = 2$ yields a 95% acceptance. As we want to have around a third of the selected facilities to be potential for mutation we set the value of $\gamma = 1$. Note that we are using a mutation rate of 1 for those genes in E_k and 0 for others. In our initial but limited experiments mutation rates of 0.1, 0.5 and 0.8 were tested and it was found that the best results were obtained when higher values are used.

Phase 2. The selection of genes for mutation

The procedure is to select a gene that is to be mutated (say x_{ki_c}, y_{ki_c}) and to exchange it with one of the fixed points which is at a certain distance away from the other facilities already chosen. A similar concept was also used by Brimberg et al. (2000) and Gamal and Salhi (2001).

Let *parent* k be a selected parent chromosome, and $\pi_k = |E_k|$ be the number of genes (points) to be mutated in *parent* k. The separation distance, r, is computed as

$$r = \min_{i \neq j} d(\boldsymbol{X}_{ki}, \boldsymbol{X}_{kj}), \quad \boldsymbol{X}_k \in \textit{parent } k,\ i, j \notin E_k,\ i, j = 1, \ldots, p.$$

The main steps of the mutation procedure are given in figure 6.

3.6. Crossover

We first choose randomly *parent* 1 from group α_1 as performed in the mutation. To obtain the second parent we perform the following tests. To produce the offspring

Step 1. Apply phase 1 for selecting the chromosome k for mutation and for constructing the set E_k that contains all the potential genes that may be mutated with *parent* k.

Step 2. For $j \in E_k$, use phase 2 to determine the fixed point to mutate point j with.

Step 3. Repeat steps 1 and 2 for K iterations to generate K children.

Figure 6. Mutation procedure.

through the genetic operator crossover, we take into account the value of the measure $\overline{N}_i$, $i = 1, \ldots, p$. We consider a facility to be attractive if it has a high $\overline{N}_i$ value. Our aim is to keep some of these attractive ones in the offspring. To perform such a task, we order the location points (genes) of the first parent in nonincreasing order with respect to the $\widehat{N}_i$ values. We then randomly choose the crossover point. Instead of selecting randomly the second parent for mating, we introduce the concept of duplication. We first order the genes of each of the other parents as we did for *parent* 1 except that we start from the smallest value this time till we reach the crossover point. We then select the first parent which has no duplication with *parent* 1, or if all parents yield some amount of duplication, we then choose the parent with the least amount of duplication.

Adopting such a strategy, we identify the first part of the offspring which has more attractive locations and the second part which has less attractive ones. Our aim is to generate new configurations that may have better fitness values. The rational behind this idea is that when two facilities are close to each other, one may dominate the other.

When applying the crossover operator the child solution produced may have better scattered locations. This observation may not always be correct as the child solution produced could be even worse than its parents if the crossover procedure creates other adjacent points in the child. The alternative child solution made from the first part of *parent* 2 and the second part of *parent* 1 can also be considered:

$$\begin{array}{rc} \textit{parent } 1\text{:} & \left[{}^{1}\overline{N}_1 \quad {}^{1}\overline{N}_2 \quad {}^{1}\overline{N}_3 \;\middle|\; {}^{1}\overline{N}_4 \quad {}^{1}\overline{N}_5\right] \\ \textit{parent } 2\text{:} & \left[{}^{2}\overline{N}_1 \quad {}^{2}\overline{N}_2 \quad {}^{2}\overline{N}_3 \;\middle|\; {}^{2}\overline{N}_4 \quad {}^{2}\overline{N}_5\right] \\ \textit{child}\text{:} & \left[{}^{1}\overline{N}_1 \quad {}^{1}\overline{N}_2 \quad {}^{1}\overline{N}_3 \;\middle|\; {}^{2}\overline{N}_4 \quad {}^{2}\overline{N}_5\right] \end{array}$$

To prevent the duplication, we introduce the following scheme for choosing the second parent:

(a) We order the location points in each solution (chromosome) in nonincreasing order with respect to $\overline{N}_i$ values, $i = 1, \ldots, p$.

(b) Given the crossover point, we choose as the second parent the chromosome whose second part has the smallest number of duplications with the first part of the first parent. Also, if a chromosome is found to have no duplication, it will be chosen as the second parent without continuing the search. Note that we could have searched for better solutions but at the expense of extra computational effort.

In case we still have some duplications in the offspring produced, we mutate the points using the mutation process described in section 3.5 so that there is no duplication in one solution. In this work we consider one third of the population size to be chosen to act as *parent* 1. The main steps of the procedure are given in figure 7.

Step 1.	Randomly select the first parent solution using the parent selection procedure of section 3.3.
Step 2.	Select the second parent as the one with the least amount of duplication.
Step 3.	If a child solution has facilities that are very close to each other, apply the mutation procedure to those similar points until distinct points are generated.
Step 4.	Repeat step 1 until the required number of child solutions is fulfilled.

Figure 7. Crossover procedure.

3.7. Chromosomes injection

This section describes a simple mechanism employed within the heuristic to maintain both solution quality and population diversity. The mechanism, which we call 'injection', is to introduce new chromosomes into the population at regular generation intervals until a prescribed number of generations is reached. This is a natural phenomenon which simulates the immigration scenario of people moving between countries or regions. This scheme proved to be effective when embedded in a GA to solve a class of VRP (see (Petch, 2001)) and continuous location–allocation problems (see (Gamal, 2001)). At the kth generation where we apply injection, the population size increases by the number of chromosomes injected and then gradually decreases to its original size.

The way we choose the chromosomes to inject into the population may affect the solution quality. One way is to choose all the chromosomes for injection randomly from the set of fixed points. Another possibility is to select some of these injected chromosomes randomly and the remaining ones from the set of good quality solutions which can be obtained by suitable heuristic techniques. In this study, we opted for the first option.

The second mechanism is the copying process. The process, whose aim is to maintain the solution quality, is simply a process of keeping the top best chromosome always in the population from one generation to the next. Figure 5 illustrates the composition of chromosomes in the new population indicating the parameters relating to both injection and copying.

4. Computational results

In this section we present the computational results obtained when using the GA-based heuristic. The proposed GA implementation is coded in Fortran 77 and compiled using f77-cg92-O4 optimiser. The execution of this heuristic is conducted on an Ultra Entreprise 450 dual processor at 300 MHz. We have used four classes of test problems given in the literature (Brimberg et al., 2000), totalling 110 instances. These four classes are based on the 50 fixed points, 287 fixed points, 654 fixed points and 1060 fixed points. The percentage deviation is based on the optimal solutions for the 50 and 287 fixed point problems and on the best known solutions for the 654 and 1060 fixed point problems (Brimberg et al., 2000). Our method is applied to these four problems using the number of open facilities, p, 2–25 for the 50-fixed point problem and 2–20 and 25–100 with an

increment of 5 for the 287-fixed point problem, 2–15 and 20–100 with an increment of 5 for the 654-fixed point problem and 5–100 with an increment of 5 for the 1060-fixed point problem mentioned above. The weights are set to unity for all problems except for the 287-fixed point problem. The deviation is computed as follows:

$$dev = \frac{F_{\text{best}} - F^*}{F^*} \cdot 100\%, \tag{2}$$

where F_{best} is the total cost found by our methods, and F^* refers to the optimal or the best found so far in the literature. The optimal solutions are found using a column generation approach combined with global optimisation and branch-and-bound (Brimberg et al., 2000).

The following parameters are set to generate computational results, these were found based on limited preliminary experiments.

- Population size, *popsize*:

$$popsize = \begin{cases} 30 & \text{for all test problems with } p \leqslant 10; \\ 50 & \text{otherwise.} \end{cases}$$

- Maximum number of generations, G:

$$G = \begin{cases} 30 & \text{for all test problems with } p \leqslant 10; \\ 60 & \text{for all test problems with } 10 < p \leqslant 50; \\ 80 & \text{otherwise.} \end{cases}$$

- Let t be the tth generation. The parameters for the parent selection and the removal procedures are:

$$\alpha_j(t) = \begin{cases} j = 1: \\ 60\% \text{ for } t \leqslant 0.4G,\ 75\% \text{ for } 0.4G < t \leqslant 0.8G,\ 90\% \text{ for } t > 0.8G; \\ j = 2: \\ 30\% \text{ for } t \leqslant 0.4G,\ 20\% \text{ for } 0.4G < t \leqslant 0.8G,\ 10\% \text{ for } t > 0.8G; \\ j = 3: \\ 10\% \text{ for } t \leqslant 0.4G,\ 5\% \text{ for } 0.4G < t \leqslant 0.8G,\ 0\% \text{ for } t > 0.8G. \end{cases}$$

- The number of chromosomes to be injected is $\max(5, 0.1 \cdot popsize)$ for all test problems, where the scheme is applied at every 5 generations, and stopped at 10 generations before the end (i.e., $G - 10$).

To obtain consistent results and to evaluate the robustness of our GA, we run the method 20 times for each instance and reported the following statistics based on the best (optimal) solutions. These include the average of the relative deviations (in %), as given by equation (2), of those 20 best results (denoted by 'Aver' in the tables). We also provide the additional information such as the standard deviation (denoted by 'Stdev'), the overall best (denoted by 'Best') and the worst (denoted by 'Worst'). In addition, we recorded the average CPU time (in secs, denoted by 'Time') of one entire run of the GA (i.e., the completion of G generations) and the average number (nearest integer) of generations required to obtain the best result, 'Gen'. We also summarised the results in the last two rows of the tables under the overall average, 'OAV', and the overall

Table 1
Summary results over 20 runs for the 50-fixed point problem.

p	Without injection						With injection					
	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #
2	0.00	0.00	0.00	0.00	0.67	8	0.00	0.00	0.00	0.00	0.69	6
3	0.00	0.00	0.00	0.00	0.76	7	0.00	0.00	0.00	0.00	0.79	5
4	0.00	0.00	0.00	0.00	0.91	12	0.00	0.00	0.00	0.00	0.97	15
5	0.00	0.00	0.00	0.00	1.00	7	0.00	0.00	0.00	0.00	1.04	12
6	0.00	0.00	0.00	0.00	1.14	20	0.00	0.00	0.00	0.00	1.19	17
7	0.00	0.00	0.00	0.00	1.19	22	0.00	0.00	0.00	0.00	1.23	21
8	0.00	0.00	0.00	0.00	1.26	15	0.00	0.00	0.00	0.00	1.33	17
9	0.00	0.00	0.00	0.00	1.31	19	0.00	0.00	0.00	0.00	1.36	19
10	0.02	0.02	0.00	0.03	1.32	16	0.01	0.01	0.00	0.03	1.35	23
11	0.00	0.00	0.00	0.00	4.37	38	0.00	0.00	0.00	0.00	4.46	45
12	0.09	0.18	0.00	0.48	4.37	24	0.12	0.26	0.00	0.62	4.42	44
13	0.00	0.00	0.00	0.01	4.44	31	0.00	0.00	0.00	0.01	4.59	33
14	0.04	0.13	0.00	0.41	4.40	32	0.00	0.00	0.00	0.01	4.54	45
15	0.10	0.22	0.00	0.65	4.47	38	0.00	0.00	0.00	0.00	4.55	35
16	0.55	0.29	0.00	0.80	4.40	40	0.59	0.26	0.09	0.94	4.49	34
17	0.69	0.54	0.03	1.49	4.33	38	0.31	0.26	0.12	0.89	4.49	32
18	0.60	0.54	0.00	1.87	4.36	44	0.31	0.33	0.01	1.08	4.47	34
19	0.10	0.06	0.00	0.14	4.27	41	0.04	0.06	0.00	0.14	4.36	51
20	0.14	0.22	0.00	0.68	4.26	41	0.06	0.09	0.00	0.28	4.33	35
21	0.07	0.18	0.00	0.57	4.26	36	0.07	0.20	0.00	0.62	4.32	30
22	0.11	0.22	0.00	0.54	4.23	43	0.06	0.17	0.00	0.54	4.35	46
23	0.05	0.11	0.00	0.27	4.22	41	0.12	0.13	0.00	0.24	4.32	48
24	0.39	0.40	0.00	0.78	4.21	31	0.30	0.36	0.00	0.78	4.35	38
25	0.75	0.60	0.00	1.91	4.16	42	0.75	0.76	0.00	1.99	4.30	28
OAV	0.15	0.15	0.00	0.44	3.10	28	0.11	0.12	0.01	0.34	3.18	29
OSTD	0.24	0.19	0.01	0.57	1.58	12	0.20	0.18	0.03	0.49	1.61	13

standard deviation, 'OSTD', for those items cited above. To highlight the impact the injection operator may have on the performance of the GA we recorded the results for both scenarios, namely, the GA with and without the inclusion of the injection operator.

Tables 1–4 produce the summary results, as described above, for the four test problems for each value of p. Our GA appears to perform extremely well, with near zero deviation, for all problems with values of $p \leqslant 20$ and starts deteriorating in the case of large values of p, ($p \geqslant 40$) especially in the larger problems. One can obviously fine tune the values of the parameters to gain more insight. In the three larger problems, the introduction of the injection operator into our GA seems to produce much better results than its counterpart besides it is shown to be more robust. The effect of the injection scheme is reflected by the higher average number of generations for which the best solution is obtained (see 'Gen' in tables). In other terms, this highlights the fact that such operator brings diversity to the population. For the case where the injection operator is not used the solution seems to stagnate at a local minimum earlier on which led to

Table 2
Summary results over 20 runs for the 287-fixed point problem.

p	Without injection						With injection					
	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #
2	0.00	0.00	0.00	0.00	9.23	1	0.00	0.00	0.00	0.00	9.25	1
3	0.00	0.00	0.00	0.00	10.69	3	0.00	0.00	0.00	0.00	11.23	10
4	0.00	0.00	0.00	0.00	11.74	4	0.00	0.00	0.00	0.00	12.45	12
5	0.00	0.00	0.00	0.00	12.19	10	0.00	0.00	0.00	0.00	12.96	12
6	0.00	0.00	0.00	0.00	10.90	11	0.00	0.00	0.00	0.00	12.33	9
7	0.01	0.02	0.00	0.08	11.91	14	0.01	0.02	0.00	0.08	12.63	9
8	0.31	0.22	0.00	0.45	13.02	16	0.05	0.14	0.00	0.43	13.48	19
9	0.13	0.21	0.00	0.45	13.70	19	0.17	0.23	0.00	0.45	14.73	14
10	0.32	0.21	0.00	0.72	13.70	18	0.35	0.15	0.00	0.59	14.49	17
11	0.23	0.12	0.13	0.48	46.04	35	0.11	0.11	0.00	0.29	47.70	29
12	0.27	0.17	0.06	0.60	48.79	20	0.19	0.16	0.00	0.62	49.62	38
13	0.46	0.31	0.06	1.05	50.65	41	0.18	0.35	0.00	1.17	52.68	46
14	0.57	0.30	0.24	1.08	58.71	37	0.40	0.34	0.00	0.89	55.41	39
15	0.50	0.38	0.00	1.40	58.86	39	0.49	0.34	0.08	1.01	58.21	38
16	0.76	0.41	0.21	1.59	62.48	34	0.62	0.53	0.00	1.68	60.72	38
17	0.86	0.75	0.04	2.10	65.46	31	0.93	0.76	0.00	2.07	64.61	46
18	1.47	0.91	0.06	2.57	68.38	42	0.57	0.50	0.06	1.55	64.37	45
19	1.39	0.74	0.35	3.09	68.47	38	1.03	0.47	0.34	1.84	65.86	51
20	1.81	0.72	0.95	2.89	71.38	35	1.10	0.60	0.21	1.72	66.61	41
25	5.71	1.76	4.01	9.09	84.17	34	1.49	0.68	0.29	2.31	77.70	44
30	10.11	3.03	5.09	14.23	93.53	19	2.50	0.75	1.69	4.01	84.93	47
35	3.16	1.16	1.69	4.90	98.18	32	3.02	1.54	0.73	5.09	96.95	39
40	5.00	1.84	2.03	7.65	101.16	35	2.13	0.99	0.16	3.62	101.37	39
45	5.10	1.84	2.98	8.72	120.64	24	2.90	0.84	1.95	4.41	113.52	39
50	14.97	6.26	6.39	25.46	136.22	29	3.44	1.19	1.71	5.55	122.65	46
55	11.15	3.19	5.65	15.08	186.17	52	4.85	1.15	2.73	6.09	170.47	45
60	34.01	10.29	9.70	43.73	232.95	27	5.73	1.24	4.53	8.53	176.85	58
65	38.44	8.24	28.56	56.03	231.28	46	6.74	0.98	4.90	7.63	179.40	53
70	56.24	8.58	39.69	68.35	244.65	55	9.93	1.63	7.82	12.54	181.18	40
75	29.03	21.35	13.90	63.17	237.80	51	9.27	1.68	6.74	12.51	182.54	53
80	52.62	31.49	15.94	92.71	265.32	38	9.14	1.30	6.84	10.58	187.76	54
85	96.75	14.89	75.18	118.02	277.61	40	9.86	1.79	7.37	13.06	190.18	45
90	107.14	14.92	91.78	128.02	286.46	42	10.34	1.99	7.59	12.78	193.60	52
95	93.35	11.03	77.59	118.96	334.21	54	10.96	1.00	8.92	12.18	195.03	58
100	135.00	14.04	108.72	151.04	353.43	34	11.01	1.85	8.59	14.84	196.81	66
OAV	20.20	4.55	14.03	26.96	114.00	30	3.13	0.72	2.09	4.29	90.01	36
OSTD	35.24	7.18	28.33	43.26	103.76	14	3.85	0.61	3.06	4.73	67.16	16

too many redundant co mputations that are wasted in the remaining generations. Such a drawback could obviously encourage the design of diversification-based schemes or introduce more randomness in the GA. No detailed statistical evidence is provided to show whether or not there is a statistically significant difference between the performance of these two GA variants.

Table 3
Summary results over 20 runs for the 654-fixed point problem.

p	Without injection						With injection					
	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #
2	0.00	0.00	0.00	0.00	14.17	1	0.00	0.00	0.00	0.00	15.95	2
3	0.00	0.00	0.00	0.00	15.35	10	0.00	0.00	0.00	0.00	16.08	5
4	0.00	0.00	0.00	0.00	15.78	5	0.00	0.00	0.00	0.00	16.81	10
5	0.00	0.00	0.00	0.00	16.75	11	0.00	0.00	0.00	0.00	17.70	12
6	0.00	0.00	0.00	0.00	18.55	3	0.00	0.00	0.00	0.00	19.98	7
7	0.00	0.00	0.00	0.00	26.27	8	0.00	0.00	0.00	0.00	25.65	12
8	0.00	0.00	0.00	0.00	23.62	13	0.00	0.00	0.00	0.00	24.56	9
9	0.00	0.00	0.00	0.00	25.26	14	0.04	0.12	0.00	0.40	28.20	17
10	0.01	0.02	0.00	0.04	25.52	14	0.01	0.01	0.00	0.02	27.84	18
11	0.00	0.00	0.00	0.00	88.36	13	0.00	0.00	0.00	0.00	93.73	6
12	0.14	0.45	0.00	1.44	92.12	26	0.00	0.00	0.00	0.00	97.34	23
13	0.20	0.49	0.01	1.60	94.15	18	0.05	0.04	0.00	0.12	98.20	33
14	0.52	0.80	0.02	2.68	98.76	30	0.08	0.18	0.01	0.60	99.98	47
15	1.29	1.05	0.56	3.09	100.16	18	0.50	0.70	0.03	1.79	102.96	48
20	6.25	1.72	4.03	9.65	112.99	15	2.61	1.56	0.48	5.07	116.27	44
25	11.55	2.26	7.43	14.58	124.92	15	4.63	1.77	1.71	7.43	136.32	45
30	14.29	3.47	8.17	21.79	141.97	5	6.31	1.10	4.21	7.85	160.38	40
35	15.48	2.87	10.62	21.02	160.21	5	8.17	1.78	6.03	12.01	184.39	42
40	14.94	1.93	12.31	18.24	173.25	6	6.05	1.83	3.40	8.93	208.40	45
45	20.84	3.09	15.40	24.02	196.70	17	9.06	2.45	5.81	12.95	243.44	48
50	23.98	2.49	19.39	28.91	229.71	6	9.47	2.60	5.00	12.38	269.35	48
55	29.30	2.99	24.75	35.66	349.98	16	13.18	0.88	11.36	14.27	410.23	57
60	38.41	2.72	35.02	43.66	357.16	12	14.64	3.15	10.62	21.12	451.67	64
65	34.64	4.45	27.91	44.35	397.84	18	15.21	2.02	11.14	17.70	485.84	61
70	33.48	2.77	30.31	38.23	422.26	10	16.20	2.62	13.06	20.11	531.14	51
75	38.32	2.63	33.65	43.03	469.32	9	17.37	2.58	12.37	20.81	573.10	60
80	40.16	4.85	31.74	46.28	491.10	10	19.52	3.32	14.07	24.06	623.33	66
85	38.30	4.53	31.78	43.49	515.95	22	19.18	2.85	15.72	23.37	668.40	59
90	41.23	2.92	37.20	45.51	596.38	13	20.00	2.81	16.39	25.08	719.03	62
95	44.19	4.63	35.32	52.55	600.35	24	21.94	3.60	14.62	25.80	770.11	62
100	42.62	5.78	36.13	55.48	688.48	52	21.55	3.33	17.63	27.33	823.41	50
OAV	15.81	1.90	12.96	19.20	215.59	14	7.28	1.33	5.28	9.33	259.99	37
OSTD	16.82	1.76	14.40	19.64	205.41	9	7.99	1.28	6.20	9.77	258.20	21

According to the results, the GA heuristic with the injection operator embedded takes about 33 mins for one run to execute the longest test problem, whereas the GA without injection requires about 45 mins. This phenomenon is mainly due to the duplications in a solution that needed to be sorted out. In other terms, the GA with injection has less duplications due to the diversity of the chromosomes within the population. Though this is comparatively much longer than the other heuristic methods used in the literature especially for the two largest problems, this needs not to be taken as a handicup as location problems generally require large investment and are of strategic nature.

Table 4
Summary results over 20 runs for the 1060-fixed point problem.

p	Without injection						With injection					
	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #	Aver (%)	Stdev (%)	Best (%)	Worst (%)	Time (secs)	Gen #
5	0.00	0.00	0.00	0.00	89.41	17	0.00	0.00	0.00	0.00	94.92	15
10	0.01	0.01	0.00	0.03	164.40	18	0.01	0.01	0.00	0.02	168.41	15
15	0.05	0.02	0.02	0.07	996.65	10	0.03	0.02	0.01	0.07	781.39	29
20	0.49	0.35	0.06	0.97	1133.83	13	0.04	0.03	0.00	0.10	849.39	48
25	0.19	0.16	0.03	0.45	1132.10	7	0.13	0.17	0.03	0.60	910.71	33
30	1.34	0.53	0.38	2.35	1084.19	9	0.47	0.38	0.01	1.07	936.58	35
35	2.62	0.58	1.69	3.17	1175.72	8	1.13	0.59	0.49	2.24	988.99	43
40	2.86	0.62	1.47	3.24	1223.93	6	1.77	0.63	1.16	3.03	1020.57	38
45	3.30	0.76	1.68	3.83	1314.71	1	2.18	0.56	1.43	3.12	1103.38	27
50	4.20	0.07	4.00	4.22	1181.45	1	2.90	0.60	2.06	3.80	1066.97	38
55	3.44	0.00	3.44	3.44	1911.62	1	3.12	0.63	1.67	3.44	1449.33	25
60	7.53	1.18	5.92	9.48	2151.26	6	3.61	1.06	1.74	4.91	1654.00	54
65	5.90	0.02	5.84	5.91	2704.97	1	4.43	0.87	2.44	5.51	1766.50	49
70	7.48	1.44	4.63	9.10	2668.41	5	5.06	1.23	3.62	7.18	1788.03	44
75	13.30	2.26	9.80	16.80	2398.89	11	5.59	0.69	4.33	6.53	1830.57	48
80	14.98	1.94	12.00	18.15	2688.17	14	5.80	0.76	4.09	7.03	1847.05	54
85	14.78	1.66	12.50	17.13	2183.04	12	5.81	0.99	4.31	7.07	1861.48	50
90	13.77	1.92	10.42	16.52	2219.13	11	6.18	0.58	5.39	7.39	1892.77	36
95	10.57	0.00	10.57	10.57	2326.93	1	5.75	0.80	4.56	6.94	1968.74	56
100	10.61	1.10	9.19	12.39	2296.07	8	6.30	0.85	4.61	7.18	1958.87	49
OAV	5.87	0.73	4.68	6.89	1652.24	8	3.02	0.57	2.10	3.86	1296.93	39
OSTD	5.24	0.74	4.41	6.22	782.77	5	2.38	0.36	1.87	2.79	569.56	12

It was also found that the best solution is usually obtained at around a third of the maximum number of generations and this information can also be used to adjust the value of G and allows probably more runs to be executed if necessary.

Comparison with the best existing GA

For completeness we have also reported the results found by the proposed GA and the best GA heuristic in the literature as presented by Brimberg et al. (2000). Table 5 summarises these findings. We denote by '$-$' a % improvement of the proposed GA against the old one and a '$+$' refers to the opposite. The bold numbers refer to both the best and the worst deviations found for each set of problem. The relative deviation (in %) is computed as $\delta = (C_{\text{new}} - C_{\text{old}})/C_{\text{old}} \cdot 100$ where C_{new} and C_{old} denote the cost generated by the new GA and the cost by the old one respectively. This expression is equivalent to $\delta = 100(a - b)/(b + 100)$ where a and b denote the deviations (in %) found by our GA and the old GA when compared to the best or optimal solution as published in the literature. According to table 5, it can be observed that for the first three data sets our GA is more robust and outperforms the existing one whereas for the last one we found slightly inferior solutions but only within 1% for both the averages and the best solu-

Table 5
Summary results of our GA vs the best existing GA.

50 customers			287 customers			654 customers			1064 customers		
p	Aver (%)	Best (%)	p	Aver (%)	Best (%)	p	Aver (%)	Best (%)	p	Aver (%)	Best (%)
2	0.00	0.00	2	−0.04	0.00	2	0.00	0.00	5	0.00	0.00
3	0.00	0.00	3	0.00	0.00	3	0.00	0.00	10	−0.02	0.00
4	0.00	0.00	4	0.00	0.00	4	0.00	0.00	15	−0.03	0.00
5	−0.01	0.00	5	−0.04	0.00	5	0.00	0.00	20	**−0.21**	**−0.08**
6	−0.04	0.00	6	−0.53	0.00	6	−0.57	0.00	25	−0.13	−0.06
7	−0.07	0.00	7	−0.20	0.00	7	−0.02	0.00	30	−0.18	−0.03
8	−0.23	0.00	8	−0.30	0.00	8	−0.05	0.00	35	0.09	0.14
9	−0.68	0.00	9	−0.21	0.00	9	0.00	0.00	40	0.40	0.39
10	−0.41	0.00	10	−0.22	0.00	10	−0.09	0.00	45	0.61	0.74
11	−1.23	0.00	11	−0.47	0.00	11	−2.10	0.00	50	0.41	0.53
12	−1.57	0.00	12	−0.63	−0.18	12	−2.06	0.00	55	1.13	0.97
13	−1.45	0.00	13	−0.43	0.00	13	−1.01	0.00	60	1.16	0.90
14	−2.71	−0.41	14	−0.41	−0.39	14	−0.44	0.00	65	**2.19**	1.47
15	−3.01	−0.16	15	−0.44	−0.09	15	−1.03	0.03	70	1.52	1.54
16	−1.10	−0.02	16	−0.78	−0.03	20	1.74	0.45	75	1.80	2.39
17	−3.54	−0.32	17	−0.30	−0.78	25	1.57	0.28	80	1.39	0.85
18	−4.07	−0.47	18	−0.66	−0.36	30	0.83	4.16	85	1.57	**2.40**
19	−5.84	−2.22	19	−0.91	−0.50	35	1.49	**2.85**	90	1.86	2.18
20	−4.17	−0.52	20	−0.48	0.14	40	−1.50	0.59	95	1.21	1.03
21	−4.61	−1.54	25	−0.82	0.28	45	**2.53**	2.56	100	1.52	0.75
22	−4.18	−0.96	30	−1.36	**1.15**	50	−0.75	−0.78			
23	−6.13	−1.79	35	−1.03	−0.01	55	0.83	2.37			
24	−7.46	−3.65	40	−3.30	**−3.87**	60	0.13	0.62			
25	**−8.23**	**−5.72**	45	−2.23	−0.63	65	−2.63	−1.27			
			50	−3.43	−2.70	70	−3.34	−1.12			
			55	−3.53	−2.87	75	−4.12	−0.11			
			60	−2.91	−1.22	80	−4.01	−2.55			
			65	−3.00	0.08	85	−4.30	−1.96			
			70	−1.77	1.10	90	−6.05	−5.72			
			75	−2.06	−0.87	95	−3.98	−4.87			
			80	−2.55	1.79	100	**−7.07**	**−6.51**			
			85	−2.25	−0.90						
			90	**−4.37**	−2.64						
			95	−2.99	−1.38						
			100	−2.87	−1.55						
OAV	−2.53	−0.74	OAV	−1.36	−0.47	OAV	−1.16	−0.35	OAV	0.81	0.81
#best	21	12	#best	33	29	# best	19	9	# best	5	3
#worst	0	0	#worst	0	5	#worst	9	9	#worst	14	14

tions. The number of better solutions found by our GA, excluding ties, as denoted by '# best', is found to be significantly large for the first three problems. The number of inferior solutions is denoted by '# worst'. It is also worth noting that when p is large, say $p \geqslant 15$, our GA outperforms even more its competitor while remaining robust in

retaining the low level of deterioration in the last data set. The execution of the old GA is performed on a Sun Sparc Station 10 and the computing time (in secs) is based on a cut off time of 100 runs of a multi-start type location method which can be found in (Brimberg et al., 2000). The recorded averages of the CPU, in secs, is approximately 3, 50, 142 and 726 secs for the 50, 287, 654 and 1060 customers problem, respectively. These figures are relatively smaller than ours for the last two data sets. It is worth noting that a direct comparison in terms of computing time may not be very informative due to the use of different machines, codes and languages adopted, among others factors.

5. Conclusion and possible research issues

A GA-based approach with a specific representation of the chromosome has been presented. Crossover and mutation operators have been designed accordingly using a new scheme for parent selection and removal strategies. In the proposed implementation of our GA, though it still has the evolutionary effect of a basic GA, some guidance is built within it to take advantage of the problem characteristics. Diversification of the search process and preservation of good quality solutions have been handled by introducing an injection policy. The proposed GA appears to outperfom the best existing GA in most instances. When compared to the best or optimal results, the obtained results are good, in general, except for large values of p, say $p > 40$, and especially for the larger data sets. In general, the GA with the population injection scheme, while not requiring extra computational effort, provides a better solution quality than the one without injection in most instances. One research issue is to adaptively adjust the parameters $(\alpha_1, \alpha_2, \alpha_3)$ based on the convergence and the behaviour of the solutions. Also the time for injection, and how many chromosomes to inject could be investigated and made to change adaptively. As a large number of chromosomes can have the same genes early on in the search it may be beneficial, from a computational view point, to design a scheme that detects this redundancy probably through a construction of some hashing functions which will then speed up the process. As the proposed GA fails for large values of p it may be worthwhile exploring ways of enhancing some of the operators that will adapt appropriately to both the size of the problem and the value of p.

Acknowledgments

The authors wish to thank the referees for their useful suggestions in an earlier version of the paper.

References

Bongartz, I., P.H. Calamai and A.R. Conn. (1994). "A Projection Method for l_p-Norm Location–Allocation Problems." *Mathematical Programming* 66, 283–312.

Brimberg, J., P. Hansen, N. Mladenović, and E.D. Taillard. (2000). "Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem." *Operations Research* 48, 444–460.

Brimberg, J. and N. Mladenović. (1996a). "Variable Neighbourhood Algorithm for Solving the Continuous Location–Allocation Problem." *Studies in Locational Analysis* 10, 1–10.

Brimberg, J. and N. Mladenović. (1996b). "Solving the Continuous Location–Allocation Problem with Tabu Search." *Studies in Locational Analysis* 8, 23–32.

Cooper, L. (1963). "Location–Allocation Problem." *Operations Research* 11, 331–343.

Cooper, L. (1964). "Heuristic Methods for Location–Allocation Problems." *SIAM Review* 6, 37–53.

Drezner, Z. (1984). "The Planar Two-Center and Two-Median Problems." *Transportation Science* 18, 351–361.

Eilon, S., C.D.T. Watson-Gandy, and N. Christofides. (1971). *Distribution Management*. New York: Hafner.

Gamal, M. (2001). "Constructive and Population Based Heuristics for the Continuous Location–Allocation Problem." Ph.D. Diss., School of Mathematics and Statistics, University of Birmingham.

Gamal, M.D.H. and S. Salhi. (2001). "Constructive Heuristics for the Uncapacitated Location–Allocation Problem." *Journal of the Operational Research Society* 51, 1233–1240.

Gamal, M.D.H. and S. Salhi. (2002). "A Cellular Type Heuristic for the Multi-Weber Problem." *Computers and Operations Research* 30.

Hansen, P., N. Mladenović, and E. Taillard. (1998). "Heuristic Solution of the Multisource Weber Problem as a p-Median Problem." *Operations Research Letters* 22, 55–62.

Hodgson, H. and S. Salhi. (2003). "A Quadtree Method to Eliminate Aggregation Error in Point to Point Allocation." *Environment and Planning B* (revised).

Houck, C.R., J.A. Joines, and M.G. Kay. (1996). "Comparison of Genetic Algorithms, Random Restart and Two-Opt Switching for Solving Large Location–Allocation Problems Problem." *European Journal of Operational Research* 20, 387–396.

Kuenne, R.E. and R.M. Solland. (1972). "Exact and Approximate Solutions to the Multisource Weber Problem." *Mathematical Programming* 3, 193–209.

Lindley and Scott. (1989). *Cambridge Statistical Tables*. London: Cambridge Press.

Love, R.F. and H. Juel. (1982). "Properties and Solution Methods for Large Location–Allocation Problems." *Journal of the Operational Research Society* 33, 443–452.

Love, R.F. and J.G. Morris. (1975). "A Computational Procedure for the Exact Solution of Location–Allocation Problems with Rectangular Distances." *Naval Research Logistics Quarterly* 22, 441–453.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evaluation Programs*. New York: Springer.

Petch, R.J. (2001). "Constructive and GA Based Heuristics for the Vehicle Routing Problem with Multiple Trips." Ph.D. Diss., School of Mathematics and Statistics, University of Birmingham.

Rosing, K.E. (1992). "An Optimal Method for Solving the (Generalized) Multi-Weber Problem." *European Journal of Operational Research* 58, 414–426.