On The Use of Genetic Algorithms to Solve Location Problems

By: Jorge H. Jaramillo, Joy Bhadury and Rajan Batta

J. Bhadury, J. Jaramillo and R. Batta "On the Use of Genetic Algorithms for Location problems" *Computers and Operations Research*, Vol. 29, 761-779 (2002). doi:10.1016/S0305-0548(01)00021-1

Abstract:

This paper seeks to evaluate the performance of genetic algorithms (GA) as an alternative procedure for generating optimal or near-optimal solutions for location problems. The specific problems considered are the uncapacitated and capacitated fixed charge problems, the maximum covering problem, and competitive location models. We compare the performance of the GA-based heuristics developed against well-known heuristics from the literature, using a test base of publicly available data sets.

Scope and purpose

Genetic algorithms are a potentially powerful tool for solving large-scale combinatorial optimization problems. This paper explores the use of this category of algorithms for solving a wide class of location problems. The purpose is not to “prove” that these algorithms are superior to procedures currently utilized to solve location problems, but rather to identify circumstances where such methods can be useful and viable as an alternative/superior heuristic solution method.

Article:

1. Introduction and motivation

Deriving optimal and near-optimal solutions to location problems has fed the growth of the field of locational analysis over the past three decades (see Daskin [1], Drezner [2] and Mirchandani and Francis [3]). However, the major models in this area, namely, the Fixed Charge Location Model, the Covering Model in Non-Competitive Location Theory and the Medianoid and Centroid Model from Competitive Location theory, are all NP-Hard, i.e. computationally difficult, combinatorial optimization problems. Hence, exact algorithms are computationally feasible for medium sized problems or for special cases (see Jacobsen [4], Krarup and Pruzan [5] and Hakimi [6]). As a result, much research effort has been devoted to devising heuristic solution procedures which run in reasonable computer time and yield solutions of acceptable quality.

Genetic algorithms (GAs) are a family of randomized-search optimization heuristics, which are based on the biological process of natural selection (see Goldberg [7]). However, applications of GAs to location models have been relatively few. In the two papers by Beasley and Chu [8] and Lorena and de Souza-Lopez [9], the authors have focussed on the application of GAs to set covering problems. For location models as such, perhaps the earliest application of GAs was in Hosage and Goodchild [10] where they presented a GA implementation for the p-median model. Recently, Owen and Daskin [11 and 12] have used GAs to solve a complex model in strategic facility location. However, to the best of our knowledge, a comprehensive study on the comparative performance of GAs on location models has not been attempted before. This is particularly significant in view of the fact that GAs have proven to be very effective on non-convex optimization problems for which it is relatively easy to assess the quality of a given feasible solution but difficult to systematically improve solutions by deterministic iterative methods. Motivated by these considerations, this paper evaluates the performance of GAs on five representative models in Location Theory. From non-competitive location models, we have chosen the Uncapacitated and Capacitated Fixed Charge Problem (see [5] and [1]) and the Maximum Covering Model (see [1] and [3]). From competitive location, we have chosen the Medianoid and Centroid models [6]. In

selecting these models, we were guided by two considerations. First, the model had to be as general as possible. Hence the problem was choice of fixed charge location rather than the p-median one, since the former includes the latter as a special case. We chose the Maximum Covering Model over the p-cover model for the same reason. Second, we wanted models for which empirical work had been performed and reported on publicly available data sets. All the five models we chose satisfied these constraints. For each of these chosen models, we develop GAs and compare their performance against well-known heuristics from the literature using a test-base of publicly available data sets.

2. Genetic algorithms: an overview

The concept of GAs was first proposed by Holland [13] and then described by Goldberg [7]. GAs are stochastic search techniques based on the mechanism of natural selection and natural genetics. GAs, differing from conventional search techniques, start with an initial set of solutions called a *population*. Each individual in the population is called a *chromosome*, and in our context, represent a solution to the problem at hand. A chromosome is a string of symbols; it is usually, but not necessarily, a binary bit string. The chromosomes evolve through successive iterations, called *generations*. During each generation, the chromosomes are *evaluated*, using some measures of *fitness*. To create the next generation, new chromosomes, called offspring, are formed by either (a) merging two chromosomes from the current generation using a *crossover* operation, or (b) modifying a chromosome using a *mutation* operator. A new generation is formed from this intermediate population by (a) selecting, according to the fitness values, some of the parents and offspring, and (b) rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the best solution converges, which hopefully represents the optimum or suboptimal solution to the problem. Let $P(t)$ and $C(t)$ be parents and offspring in current generation t; the general structure of GAs is illustrated in Fig. 1 and described as follows:

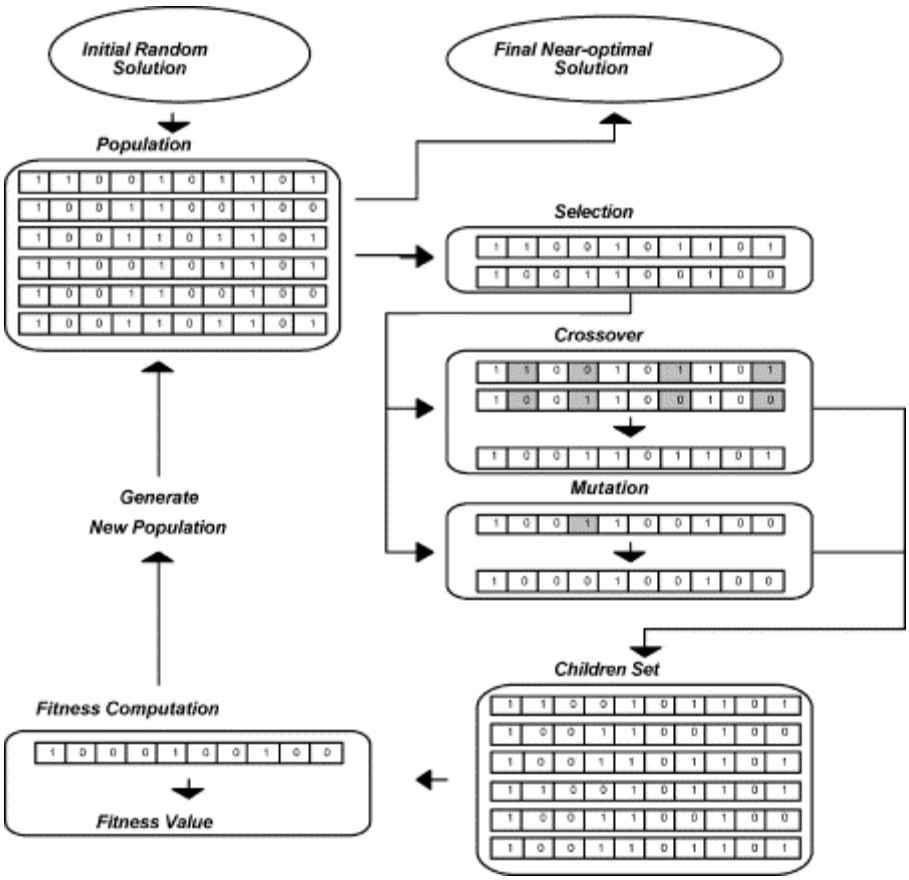

Fig. 1. The general structure of genetic algorithms.

Step 1: Set $t \coloneqq 0$.
Step 2: Generate initial population, $P(t)$.

Step 3: Evaluate $P(t)$ to create fitness values.
Step 4: *While* (not termination condition) *do*.
Step 5: Recombine $P(t)$ to yield $C(t)$, selecting from $P(t)$ according to the fitness values.
Step 6: Evaluate $C(t)$.
Step 7: Generate $P(t + 1)$ from $P(t)$ and $C(t)$.
Step 8: Set $t := t + 1$.
Step 9: *End*.
Step 10: Stop.

We used two termination criteria in implementing the above generic GA. This basic algorithm was executed until one of the following happened: either the GA "converged" (i.e., improvements in objective function value fell below tolerance limit of 10^{-5}) or it had been executed for a pre-specified number of interventions. The reader is referred to references [7 and 14] and Reeves [15] for a more detailed coverage of GAs.

3. GA-based heuristics for location problems that we studied

The basic GA scheme permits a wide variety of ways of implementing a GA-based heuristic, and exploring these has produced a large body of scientific literature. In this section, we shall describe in brief the major modifications that we made to the basic GA, and the criteria adopted from the literature to obtain a suitable GA-based heuristic for the location problems we considered. This is intended only to be a brief overview; the reader is referred to [16] for complete details.

3.1. Representation scheme

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme. The representation scheme developed was a n_f-bit binary string as the chromosome structure, where n_f is the number of potential facility sites. A value of 1 for the ith bit implies that a facility is located in the ith site. The binary representation of an individual's chromosome (solution) is illustrated in Fig. 2.

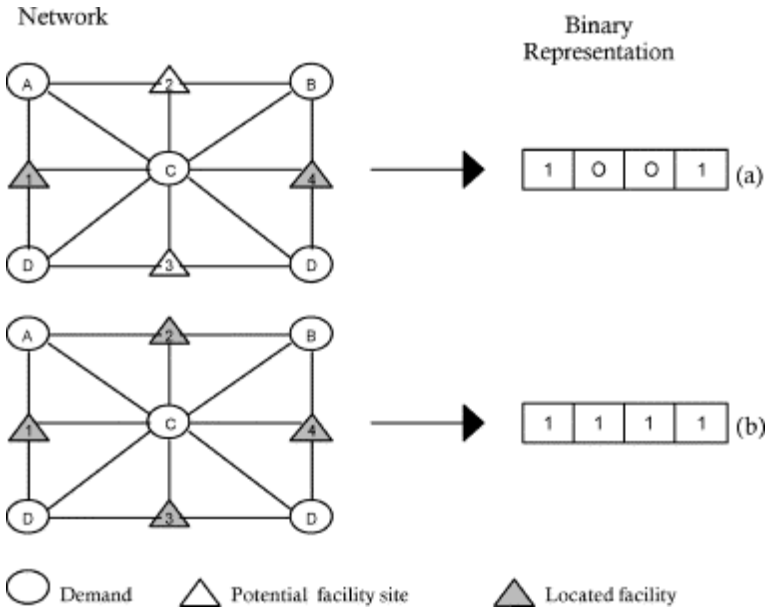

Fig. 2. Example of binary representation for location problems.

The network illustrated in Fig. 2 has four potential facility sites. The case (a) represents the situation when only two facilities are located in potential sites one and four and case (b) illustrates the situation when facilities are located in all of the possible facility sites. The respective binary representations for cases (a) and (b) is also shown in the same figure.

3.2. Fitness function

The fitness function in a GA is a measure of goodness of a solution to the objective function. In our modified GA, the fitness of an individual is directly related to its objective function value.

3.2.1. Fitness function of the uncapacitated fixed charge problem

The objective function of the uncapacitated fixed charge location problem can be formulated as follows:

$$\sum_j f_j X_j + c \sum_i \sum_j h_i d_{ij} Y_{ij}.$$

where

$$X_j = \begin{cases} 1 & \text{if we locate at candidate site } j, \\ 0 & \text{otherwise,} \end{cases}$$

Y_{ij} = fraction of demand at node i that is served by a facility at node j,
f_j = fixed cost of locating at candidate site j,
h_i = demand at node i,
d_{ij} = distance from demand node i to candidate location j,
c = cost per unit distance per unit demand.

The fitness of an individual can be calculated by evaluating the two components of the objective function (fixed facility cost and transportation cost) and then adding them.

The fixed facility cost for solution i is calculated by

$$ffc_i = \sum_{j=1}^{n} f_j s_{ij}$$

where s_{ij} is the value of the jth bit in the string corresponding to the ith individual. In order to evaluate the transportation cost, we first find the set of open facilities given the current solution. Next, since the facilities are uncapacitated, we compute the respective minimum transportation cost by assigning each demand node to the nearest open facility.

3.2.2. Fitness function of the capacitated fixed charge problem

The objective function of the uncapacitated fixed charge problem is identical to the UFC problem. The fixed facility cost is calculated exactly as in the UFC problem. In order to calculate the transportation cost, we note that if we are given a set of facilities that are feasible in the sense that the total capacity of these facilities exceeds the total demand, the problem of assigning the demands to the facilities becomes a transportation problem which can be solved using special algorithms such as transportation simplex method. When the capacity of the facilities is less than the total demand, the current solution is not feasible and a penalty value is assigned to the fitness of the individual. We assigned a numerical value considerably larger than any possible objective function value corresponding to the current population of individuals.

3.2.3. Fitness function of the maximum covering problem

The fitness function value of each possible solution to the MC problem is calculated by finding the set of open facilities belonging to the solution being evaluated and adding the demands that are covered by these facilities. Care must be taken in order to avoid demands being counted several times since one demand point might be covered by several facilities at the same time.

3.2.4. Fitness function of the medianoid and centroid problem
Medianoid problem: The fitness of each possible solution to this problem is calculated by finding the market share of the set of open facilities belonging in this case to the follower. The market share is found by assigning each demand node to the nearest open facility and adding only those demands falling into the set of facilities of the follower.

Centroid problem: The (r |p)-centroid problem is essentially a minmax problem faced by the player (the leader). For every candidate solution considered by the leader, it is necessary to compute or approximate the follower's best response, and this requires either solving a (r | X_p)-medianoid problem or estimating an approximate response through the greedy algorithm described in Benati and Laporte [17]. The total value of the demands minus the demands covered by the follower is the fitness of the individual being evaluated.

3.3. Parents selection-procedure
We will now address the issue of selection of parents, i.e., solutions chosen for crossover. For this we chose the Binary Tournament Selection Method (see Beasley and Chu [8]) because (i) it can be implemented very efficiently, and (ii) in Beasley and Chu [8] it has been shown that this method gives solutions whose quality compare favorably to the ones produced by other methods.

3.4. Crossover operator
We chose the crossover operator fitness-based fusion proposed in Beasley and Chu [8]. This operator is more capable of generating new solutions when two parent solutions are similar in structure than the one-point crossover operator, since it focuses on the differences in the two structures. However, the possibility of getting an offspring identical to one of its parents still remains. As a result of this problem, the following modification was made to the overall crossover operator.

Step 1: Select parents P_1 and P_2.
Step 2: Apply the fusion operator to obtain an offspring (C_1)
Step 3: Compare Child C_1 with its parents. If it is not identical to either then go to Step 5. Otherwise go to step 4.
Step 4: Apply *mutation* to the parent with the lesser fitness.
Step 5: Stop.

3.5. Mutation operator
Mutation is a process that reverses the structure of a chromosome and hence produces albinos, i.e., individuals with different chromosome properties from the majority in a population. Mutation serves as a policy to prevent solutions from being trapped in local optima and is considered as a secondary mechanism in the operation of genetic algorithms. In this work, the mutation operator works by selecting randomly one of the open facilities and moving this to another site. The new site is also picked randomly from the set of empty possible places to locate facilities.

Mutation rate (probability) is usually set to a very low level. However, different references [8 and 14] have found that a higher mutation rate is necessary when the GA has converged. In order to alter the mutation rate when the GA is in progress, Beasley and Chu [8] made use of a variable mutation rate rather than a fixed one and this variable rate depends on the rate at which the GA converges. The approach employed here uses mutation during the progress of the GA and the crossover operator is substituted by the mutation operator in those cases when the offspring is identical to one of its parents.

3.6. Replacement population method
Once new child solutions have been constructed through the GA operators, the child solutions will replace “less fit” members of the population. The average fitness of the population will improve if the child solutions have better fitnesses than those of the solutions being replaced. This type of method is called “incremental replacement”. Another commonly used method is the “generational replacement”, which generates a new population of children and replaces the whole parent population (see Beasley et al. [18]).

In our GA, we used the incremental replacement method. Using this method, the best solutions are always in the population and the newly created solutions are immediately available for selection and reproduction. Note that when replacing a solution, care must be taken to prevent excessive copies of a solution from entering the population. Allowing too many duplicate solutions to exist in the population may be undesirable because a population could come to consist of identical solutions, thus severely limiting the GAs ability to generate new solutions.

3.7. Population size

One of the most obvious questions relating to GA performance is how it is influenced by population size. In principle, it is clear that small populations run the risk of seriously under-covering the solution space, while large populations incur severe computational penalties. The experimental work by Alander [19] suggests that a value between n and $2n$ is optimal for the problem type considered, where n is the length of a chromosome. We chose a value of n and in our situation the length of a chromosome coincided with the number of possible facility sites.

3.8. Algorithm

Thus, the GA that we implemented can be described algorithmically as follows:

Step 1: Set $t \coloneqq 0$.
Step 2: Generate initial population, $P(t)$ randomly.
Step 3: Evaluate each of the strings in $P(t)$ according to the kind of problem being solved.
Step 4: While (number of generations $\leq$ maximum value) or (improvement in objective function value $\leq 10^{-5}$) do.
Step 5: Set $t \coloneqq t + 1$.
Step 6: Select two solutions P_1 and P_2 from the population using binary tournament selection.
Step 7: Apply genetic operators to strings P_1 and P_2.
    If Crossover: Combine P_1 and P_2 to form a offspring O_1 using the fusion crossover operator. If O_1 is identical to any of its parents, then apply mutation operator to the parent with the best fitness.
    If Mutation: Apply mutation operator to the parent with the best fitness to form a offspring O_1.
Step 8: Repeat steps 6 and 7 until a new set of children is created which is of same size as the parent population.
Step 9: Evaluate this new child set according to the kind of problem being solved.
Step 10: Utilize the incremental replacement method to create $P(t)$ from the parent population and set of children.

4. Computational results

The heuristics were coded in FORTRAN 90 and executed on a PC COMPAQ Presario 4716. Initial tests of comparison between a Sun Sparc workstation and the PC showed the latter to be faster. This PC was equipped with 32 MB of RAM and a Intel Pentium microprocessor running at 200 MHz. Two different tests were applied to each heuristic. First, the performance of the heuristic was compared against that of other heuristics achieved in the past. Second, in order to see the characteristics of convergence of each heuristic proposed and its performance on real-life problems, each heuristic was used to solve a real-world problem. This real-world problem is a network that models Amherst, a suburb of Buffalo, New York. This network has 84 potential facility sites and 459 demand nodes.

The GA-based heuristics were run on each problem multiple times since GAs are stochastic and therefore yield different searches and potentially different results, for each random number seed used. Ten different seeds were selected to create 10 genetic runs for each problem. These 10 runs tested the dependence of the method on the seed selected, both in terms of solution quality and search effort. When the solution quality showed little sensitivity to the choice of the initial seed, we considered the GA to be "*robust*".

4.1. Uncapacitated fixed charge problem

For the UFC problem, 15 problems were solved. For all of these test problems, the optimal solution is known. These test problem sets are publicly available electronically from *http://mscmga.ms.ic.ac.uk/info.html.* The results obtained were compared in terms of solution quality and solution time to those obtained by the Lagrangean heuristic implemented by Beasley [20]. Beasley's heuristic was selected since the computational experience reported shows that it is robust and efficient. In addition, the performance of these heuristics on these publicly available data sets is also well documented.

Table 1 summarizes the experimental results on the set of test problems. In this table, we list the problem name, the number of potential facilities (n_f), the number of demand sites (n_d), the optimal solution value, the percentage of times that the GA-based heuristic reached the optimal value, the average CPU solution time, the percentage of deviation from optimal for the best solution found by the Lagrangean heuristic, and the computation time for the Lagrangean heuristic.

Table 1. Comparison between the GA and the Lagrangean heuristic for the UFC problem

Beasley problem	n_f	n_d	*Optimal value*	GA		Lagrangean	
				% of times optimal sol. reached	*Mean CPU time (s)*	% deviation from optimal	*CPU time (s)*
VII-1	16	50	932 615.8	100	0.027	0	0.11
VII-2			977 799.4	100	0.033	0	0.08
VII-3			1 010 641.6	100	0.033	0	0.11
VII-4			1 034 977.1	100	0.026	0	0.05
X-1	25	50	796 648.4	100	0.143	0	0.18
X-2			854 704.2	100	0.081	0	0.16
X-3			893 782.1	100	0.115	0	0.14
X-4			928 941.8	100	0.044	0	0.11
XIII-1	50	50	793 439.6	100	0.605	0	0.31
XIII-2			851 495.4	100	0.625	0	0.28
XIII-3			893 076.8	100	0.467	0	0.29
XIII-4			928 941.8	100	0.286	0	0.15
A	100	1000	17 156 456.0	100	23.31	0	2.2
B			12 979 069.0	100	60.129	0	4.43
C			11 505 597.0	100	76.226	0.033	4.45

The following remarks can be made regarding the results illustrated in Table 1.

- The GA-based heuristic reached the optimal solution for all 150 problems unlike the Lagrangean heuristic which failed to find the optimal solution for the last problem C. The solution obtained by the Lagrangean heuristic was 0.033% above the optimal.
- The average running times of the GA heuristic are lower than those presented by the Lagrangean heuristic when “small-sized” problems were solved. The opposite situation is observed when “large-sized” problems were tackled by both heuristics. However, this observation should be viewed with caution in light of the fact that (i) the hardware used by the Lagrangean heuristics was different, and (ii) location problems are usually strategic and are not solved repeatedly.

The GA-based heuristic was also used to solve a UFC problem for the real-life Amherst network described earlier. In this case, the cost of building a facility in the region modeled had to be assumed. Borrowing from Daskin [1], current statistics about the construction business around this area were consulted and a value of US $81,725 per facility was used. Table 2 illustrates the experimental results when 10 different seeds were selected to create 10 genetic runs of the same problem. The algorithm reached the same objective function value in all of the instances. These results indicate that GA-based heuristics are robust.

Table 2. Results of GA for the UFC problem, for the Amherst problem

Case	Objective function value	CPU time (s)
1	2 568 942.5	46.14
2	2 568 942.5	41.41
3	2 568 942.5	40.92
4	2 568 942.5	23.34
5	2 568 942.5	11.32
6	2 568 942.5	23.34
7	2 568 942.5	24.06
8	2 568 942.5	51.35
9	2 568 942.5	27.14
10	2 568 942.5	13.23

4.2. Capacitated fixed charge problem

For the CFC problem, 26 problems also taken from the web site *http://mscmga.ms.ic.ac.uk/info.html* were solved. For all of these test problems, the optimal solutions are available at this web site. Our experimentation revealed that while the GA heuristic was able to reach the optimal solution in all cases but one, the computational times were excessively large. In fact, the solution times were so high that only five seeds were used for each run. As a result of this unsatisfactory performance, we did not pursue this avenue further. The reader is referred to Jaramillo [16] for details.

4.3. Maximum covering problem

Two sets of data were used to test the GA-based heuristic. The first was obtained from the literature. This problem corresponds to the 88-cities problem defined by Daskin [1]. The 88 cities selected in this problem are the 50 most populous ones in the lower states according to the 1990 Population and Housing Census as well as capitals of the lower 48 states. The second data set was obtained by modifying the one from the literature. This modified data set is a 150-node network obtained by adding 62 new nodes to the 88-cities problem and generating distances and demands randomly. The results obtained were compared to those obtained using the program referenced in this paper as SITATION which comes with a location theory text by Daskin [1]. SITATION uses a Lagrangean heuristic followed by a substitution procedure. This substitution procedure takes a given solution and attempts to improve it using a greedy heuristic — for details refer to Daskin [1]. In order to make the comparison unbiased, we implemented the substitution procedure after the GA heuristic as well. Detailed computational results are shown in Table 3. In this table, we list the problem name, the number of potential facilities (n_t), the number of facilities to be located (X_p), the distance beyond which a demand is considered uncovered (coverage distance), the percentage of the total population covered (best solution), the percentage of times that the GA-based heuristic reached the best solution value, the percentage of deviation from the best solution for the worst answer found by the GA-based heuristic, the average CPU solution time, the percentage of deviation from the best solution for the best answer found by the Lagrangean heuristic, and the computation time for the Lagrangean heuristic. The optimal solutions to these problems were not identified. Consequently, the best solution value is defined as the best solution found between the two heuristics. The conclusion we can draw from these results is that the GA method appears to give results that are at least as good as the Lagrangean method, though it does take significantly more computational effort in doing so. The improvement in results will be significant in applications where large budgets were involved (e.g., 0.05% of $500 M is significant). It is also noted that these results are a representative set from a sample of runs. More details can be found in Jaramillo [16].

Table 3. Comparison between the GA followed by substitution and the Lagrangean heuristic followed by substitution

Problem	n_i	X_p	Coverage distance (miles)	*Best solution* (%)	GA			Lagrangean	
					% of times best solution reached	% deviation from best solution (worst case)	*Mean CPU time* (*s*)	*% deviation from best solution*	*CPU time* (*s*)
City 88-1	88	2	720	89.0	100	0	0.16	0	0.06
City 88-2		3		100.0	100	0	1.163	0.05	0.25
City 88-3		2	410	61.1	100	0	0.089	0	0.04
City 88-4		3		78.2	100	0	0.908	0	0.05
City 88-5		4		87.5	100	0	0.472	0	0.06
City 88-6		5		92.5	100	0	3.208	0	0.08
City 88-7		6		96.7	100	0	3.932	0	0.07
City 88-8		7		99.8	100	0	4.411	0	0.08
City 88-9		8		100.0	100	0	5.014	0	0.14
City 150-1	150	3	720	83.6	100	0	7.102	0	0.19
City 150-2		5		94.5	100	0	11.748	0.08	0.22
City 150-3		6		97.8	100	0	14.902	0	0.18
City 150-4		7		100.0	100	0	16.451	0	0.19
City 150-5		4	410	56.5	100	0	0.978	0	0.07
City 150-6		8		77.5	100	0	28.298	0	0.12
City 150-7		12		89.0	100	0	41.712	0	0.15
City 150-8		16		95.1	100	0	70.572	0	0.17
City 150-9		20		98.9	100	0	64.087	0	0.15
City 150-10		21		99.5	100	0	59.011	0	0.17
City 150-11		22		100.0	100	0	53.984	0	0.17

As we can conclude from Table 3, the GA heuristic follow by substitution to solve the MC problem is relatively expensive computationally when compared with the Lagrangean heuristic. However, the quality of the solutions found by the GA Heuristic is better. The GA heuristic with the substitution procedure found the best solution in all of the 200 trials. On the other hand, the Lagrangean heuristic with substitution failed to find the best known solution in two out of 20 problems.

Table 4 illustrates the results when the GA-based heuristic (with substitution) was used to solve a real-life MC problem using the Amherst network. Once again, the results indicate a high degree of robustness. The results are relevant since using the substitution procedure appears to always make the GA method superior to the Lagrangean heuristic with substitution. It is, therefore, recommended that the GA-based heuristic be followed by the substitution procedure. Again, the results are a representative set from a sample of runs, with details found in Jaramillo [16].

Table 4. Results of Amherst data set (GA heuristic followed by substitution procedure)

Problem	n_d	n_1	X_p	Coverage distance (miles)	*Best solution* (%)	GA	
						% of times best solution reached	*Mean CPU time* (*s*)
Amhe-1	459	84	5	40	98.6	100	2.598
Amhe-2			6		99.7	100	9.535
Amhe-3			7		99.8	100	5.882
Amhe-4			8		99.9	100	6.674
Amhe-5			9		100.0	100	2.624

4.4. Competitive location problems

Even though there are several heuristics reported in the literature to solve the medianoid and centroid problem, the computational experience and the test problems available are still very restricted. For example, although Benati and Laporte [17] have a good heuristic for competitive location models, their data sets are not available for direct comparison or benchmarking. Consequently, we used the well-known 55-node Swain network [21] utilized by Serra and ReVelle [22] and developed two additional test problems based on this network. The size of these two problems was limited to 30 and 37 nodes in order to find the optimal objective function value by enumerating all possible solutions.

4.4.1. Medianoid problem

Table 5 summarizes the experimental results on the set of the test problems. In this table, we list the problem name, the number of demand sites (n_d), the number of potential facilities (n_f), the number of existing facilities (X_p), the number of facilities to be located by the follower (Y_r), the optimal solution value, which corresponds to the demand attracted by the new facilities (follower), the percentage of times that the GA-based heuristic reached the optimal value, and the average CPU solution time. These results show that the GA-based heuristic found the optimal solution for all 40 trials solved. This heuristic is also notably efficient in terms of execution time.

Table 5. Solutions obtained with the GA and an exact algorithm

Problem	n_d	n_f	X_p	Y_r	Optimal value	GA	
						% of times optimal solution reached	*Mean CPU time (s)*
P30-1	30	30	3	3	1158	100	< 0.001
P30-2			4	3	867	100	0.022
P37-1	37	37	2	2	1324	100	< 0.001
P37-2			3	3	1308	100	0.033

Table 6 illustrates the results when the GA-based heuristic was used to solve a real-life medianoid problem using the Amherst data set. In this table, we list the case name, the number of demand sites (n_d), the number of potential facilities (n_f), the number of existing facilities (X_p), the number of facilities to be located (Y_r), the best solution value, which coincides with the demand attracted by the new facilities (follower), the percentage of times that the GA-based heuristic reached the best solution value, the percentage of deviation from the best solution for the worst answer found by the GA-based heuristic, and the average CPU solution time. These results show that the GA-based heuristic gave the same answer 31 times out of 40, and the maximum difference between the worse solution and the best one was only 0.5% of the total demand. These results indicate a high degree of robustness. It must be pointed however that such large-scale medianoid problems could also be solved using a commercial solver and Revelle's integer programming for mutations [15].

Table 6. Results for Amherst data set

Problem	n_d	n_f	X_p	Y_r	Best solution	% of times best solution reached	*% deviation from best solution (worst case)*	*Mean CPU time (s)*
1	459	84	5	5	61 730	100	0	9.27
2			10	10	58 985	80	0.2	16.61
3			15	15	56 306	70	0.4	26.3
4			20	20	58 288	60	0.5	39.3

4.4.2. Centroid problem

As is evident from the problem description, the centroid problem necessitates the repeated solving of associated medianoid problems. These medianoid problems may be solved using either the greedy algorithm suggested by Benati and Laporte [17] or by using the GA heuristic we developed in the earlier section. This gives rise to two different GAs for the centroid problem itself. The variant in which the greedy algorithm is used as a subroutine to solve the medianoid problems is referred by us as the *GA Greedy heuristic*. On the other hand, if the GA heuristic of the previous section is used, the resulting procedure will be referred to as the *regular GA heuristic* for the centroid problem. Our initial experimentation reveals that, the GA greedy heuristic is as good as the regular GA heuristic in terms of quality of solutions. However, in terms of running time, the GA greedy heuristic was about six times faster than the GA one. Given this empirical finding, we performed the remaining evaluations using only the GA greedy heuristic.

Table 7 illustrates the results obtained when the GA greedy heuristic and the one developed by Serra and ReVelle [22] were used to solve the same centroid problem on the 55-node Swain network [21]. In this table, we list the number of facilities to be located by the leader (X_p), the number of facilities to be located by the follower (Y_r), the lowest, average and highest final capture (absolute) obtained by the leader after ten runs, and the average CPU solution time.

Table 7. GA greedy heuristic against Serra and ReVelle's heuristic

X_p	Y_r	GA greedy				Serra and ReVelle			
		Final capture				Final capture			
		Low	Mean	High	*Mean CPU time (min)*	Low	Mean	High	*Mean CPU time (min)*
1	1	1697	1697	1697	0.00	1697	1697	1697	0
2	2	1705	2705	1705	2.02	1705	1705	1705	13.7
3	3	1631	1631	1631	2.82	1498	1591	1631	19.8
4	4	1671	1674	1677	3.71	1623	1651	1677	26.9
5	5	1591	1633	1648	4.66	1591	1627	1648	39.9
6	6	1611	1617	1627	6.70	1594	1612	1627	47.9
7	7	1595	1612	1621	8.15	1555	1594	1616	51.7
8	8	1582	1624	1644	9.53	1627	1637	1644	62.4
9	9	1636	1655	1673	11.01	1652	1662	1673	73.5

The following remarks can be made regarding the results illustrated in Table 7.

- The GA greedy heuristic is notably more efficient than the Serra and ReVelle's heuristic in terms of execution time.
- The GA greedy heuristic improved by 0.3% the best solution found by Serra and ReVelle's heuristic when the leader and the follower located seven facilities, respectively. The solution of 1621 was found when the leader located its facilities on the sites 4, 5, 7, 10, 22, 36 and 38.
- In comparison with the Serra and ReVelle's heuristic, the GA greedy heuristic obtained better or equal mean solutions in seven out of nine problems.

Table 8 illustrates the results when the GA-based heuristic was used to solve a real-life centroid problem using the Amherst data set. Four different problems were constructed modifying the number of facilities to be located by the leader and follower, respectively (5, 10, 15, 20). Each problem was solved five times in order to observe the capacity of the heuristic to arrive at the same solutions. These results indicate a high degree of robustness since the highest deviation of the demand taken by the leader corresponds to 0.46% of the total demand. Further, to the best of our knowledge, this is the largest size application of the Centroid Model reported in the literature.

Table 8. Results for the centroid problem-using Amherst data

Case	5–5		10–10		15–15		20–20	
	Final solution	CPU time (s)	Final solution	CPU time (s)	Final solution	CPU time (s)	Final solution	CPU time (s)
1	55 268	1321	59 169	3326	62 152	6008	64 764	7568
2	56 170	1312	59 861	3313	62 725	6001	65 122	7373
3	56 170	1295	59 843	3296	61 913	5866	65 439	7364
4	56 170	1307	60 110	3317	61 978	5916	65 593	7483
5	56 170	1306	60 773	3335	62 725	5992	65 624	7182
Average	55 990	1308	59 951	3318	62 299	5957	65 308	7394
Deviation	360,6	8,45	517	13,05	356,92	56,25	325,26	130

4.5. GA operators

Mutation rate is usually set to a very low level as mentioned in Section 3.5. However, the GA developed to solve location problems uses a dynamic mutation rate. Fig. 3 illustrates the mutation rate observed when the GA heuristic solved one of the problems reported in the previous sections. It can be seen that at the beginning the crossover operator is mainly responsible for the search. As the GA progresses, the mutation rate becomes more productive and so the crossover rate decreases. It is noted here that, in most cases, by the time this change

occurs (i.e., the mutation rate becomes more productive) the GA has already achieved very decent solution quality. So even though the mutation rate becomes more productive, the improvement in solution quality from this point on is marginal.

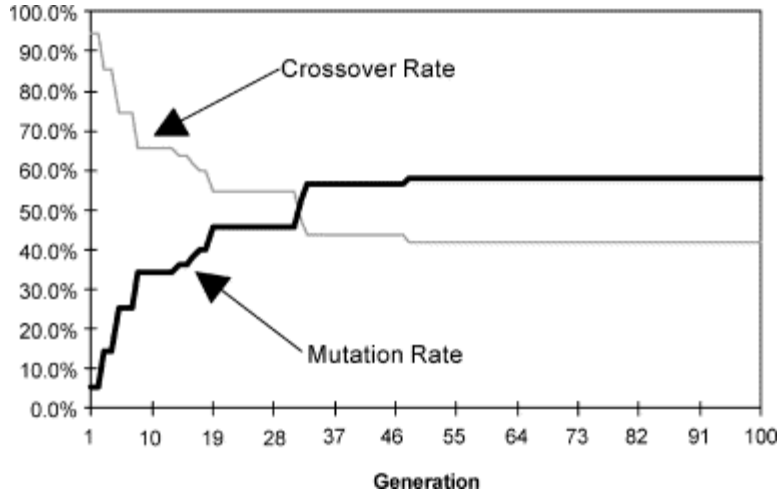

Fig. 3. Crossover and mutation rate.

Fig. 4 illustrates one of the advantages of the replacement population method adopted in our GA. In this case, when the GA heuristic stops, we have available not only the best solution but also a set of different good solutions, all of them with just a modest percentage away from the best solution.

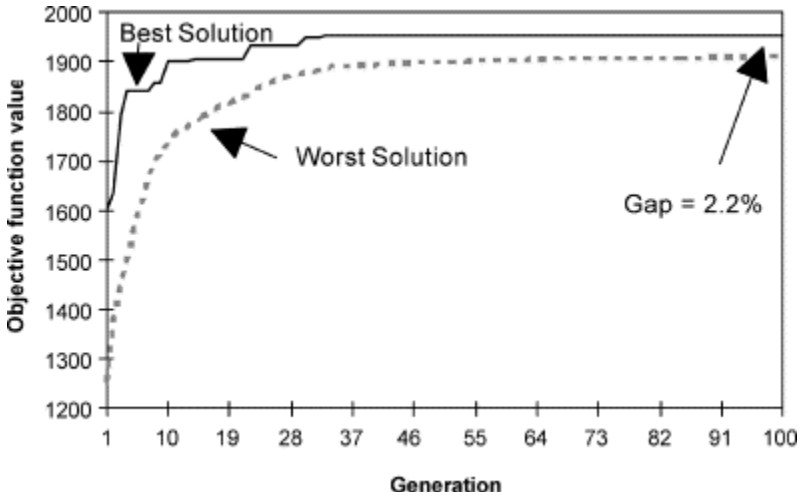

Fig. 4. Evolution of the population.

Finally, we present the results of another test that we ran in order to investigate the rate of improvement of the objective function value while using GA. For this, UFC problems were chosen from the same publicly available data set mentioned in the paper, where optimal objective function values are also given for each problem. We ran our GAs for each of these problems under two separate termination criteria: run until the optimal solution was obtained or, stop when within 1% of the optimal. The results are shown in Table 9. In this table, we list the problem name, the average running time when only optimal solutions were accepted, the average running time when solutions are within 1% of optimal solution were allowed, and the percentage of time's reduction when the second policy was adopted. These results indicate that the GA heuristic developed spends about 50% of its total consuming time trying to improve solutions that are already 1% or less over the optimal solution.

Table 9. GAs running time

Beasley problem	*Optimal*	*% away*	% reduction in time
	Mean CPU time (s)	*Mean CPU time (s)*	
VII-1	0.027	0.012	54
VII-2	0.033	0.014	57
VII-3	0.033	0.016	49
VII-4	0.016	0.006	62
X-1	0.143	0.088	38
X-2	0.081	0.038	53
X-3	0.115	0.06	47
X-4	0.044	0.022	51
XIII-1	0.605	0.261	57
XIII-2	0.625	0.224	64
XIII-3	0.467	0.209	55
XIII-4	0.286	0.198	31
A	23.31	16.7832	28
B	60.129	25.838	57
C	76.226	32.742	57

5. Conclusions and further research

This paper represents the first attempt to apply the technique of GAs to solve a comprehensive set of problems in location theory. The five representative location models that were chosen are the fixed charge location problem (uncapacitated and capacitated version) and the maximum covering problem from non-competitive location theory and the centroid and medianoid models from competitive location theory. Our criteria in choosing these models were (i) ensure that they were as general as possible, (ii) well-known heuristics were publicly available for solving them, and or (iii) performance of these heuristics on publicly available data sets was well documented. GAs were developed for each of these models and tested extensively on readily available hardware for benchmarking.

The primary conclusion from our study is that GAs demonstrate a mixed performance in solving these four classical location models above. Overall, our testing shows that for the first three models, GAs tend to take a lot more time than specialized heuristics. However, the up-side is that the solutions that the GAs produce are no worse than and in fact, sometimes superior to the ones produced by these other methods that are available in the literature. As for excessive computational times, they may be rationalized in view of the fact that locational decisions by firms are mostly strategic in nature; hence, in almost every application, location models do not need to be solved on a repeated basis. The other encouraging feature about GAs is that our limited testing on the UFC reveals that they quickly evolve to give a "good" solution; however, having obtained a good solution, GAs spend an excessive amount of time in getting marginal improvements. When it comes to capacitated fixed charge models, our finding is that GAs perform very poorly and should not be adopted.

As for competitive location models, GAs seem to perform well with regards to both computation time and solution quality. In fact, to the best of our knowledge, the largest centroid model ever solved by a heuristic as reported in the published literature is in this paper.

GAs were also tested on a large, real-life data set that was constructed based on the road network in Amherst, New York. Our testing revealed that GAs were able to give good solutions for each model (except the capacitated fixed charge location problem) and in addition, solution quality was quite robust with respect to initial starting conditions of the GA.

However, it must be kept in mind that these findings are the results of a preliminary study. Much more work needs to be done before determining the suitability or even superiority of GAs as a viable solution procedure for location models. Therefore, the most immediate avenue for future research should be to conduct additional empirical work on GAs on other location models such as the *p*-median model, the center model, etc. Further, it may be worthwhile to benchmark the performance of GAs against other metaheuristics such as Tabu Search and/or Simulated Annealing algorithms for some of these models. Finally, another strand of future research could investigate the development of "specialized" GAs for each different location model.

References:

1. M.S. Daskin. *Network and discrete location: models, algorithms, and applications*, Wiley, New York, NY (1995).
2. Drezner Z, editor. Facility location: a survey of applications and methods. New York, NY: Springer, 1995.
3. Mirchandani PB, Francis RL, editors. Discrete location theory. New York, NY: Wiley, 1990.
4. S.K. Jacobsen, Heuristics for the capacitated plant location model. *European Journal of Operational Research* **12** (1983), pp. 253–261.
5. J. Krarup and P.M. Pruzan, The simple plant location problem: survey and synthesis. *European Journal of Operational Research* **12** (1983), pp. 36–81.
6. S.L. Hakimi, Locations with spatial interaction: competitive locations and games. In: P.B. Mirchandani and R.L. Francis, Editors, *Discrete location theory*, Wiley, New York, NY (1990).
7. D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, MA (1989).
8. J.E. Beasley and P.C. Chu, A genetic algorithm for the set covering problem. *European Journal of Operational Research* **94** (1996), pp. 392–404.
9. L. Lorena and L. deSouza-Lopez, Genetic algorithms applied to computationally difficult set covering problems. *Journal of the Operational Research Society* **48** (1977), pp. 440–445.
10. C.M. Hosage and M.F. Goodchild, Discrete space location-allocation solutions for genetic algorithms. *Annals of Operations Research* **6** (1986), pp. 35–46.
11. Owen SM, Daskin MS. Strategic facility location via evolutionary programming. Working paper, Dept of Industrial Engineering and Management Science, Northwestern University, Evanston, Illinois, 1998.
12. Owen SH, Daskin MS. A note on evolution programs for solving multi-objective strategic facility location problems. Working Paper, Dept of Industrial Engineering and Management Science, Northwestern University, Evanston, Illinois, 1998.
13. J.H. Holland. *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor (1975).
14. M. Gen and R. Cheng. *Genetic algorithms and engineering design*, Wiley, New York (1996).
15. C.R. Reeves. *Modern heuristic techniques for combinatorial problems*, Wiley, New York (1993).
16. Jaramillo JH. Genetic algorithms for location problems. Unpublished MS thesis, Department of Industrial Engineering, State University of New York at Buffalo, Buffalo, NY 14260.
17. S. Benati and G. Laporte, Tabu search algorithms for the ($r \mid X_p$)-medianoid and ($r \mid p$)-centroid problems. *Location Science* **2** (1994), pp. 193–204.
18. J.E. Beasley, D.R. Bull and R.R. Martin, An overview of genetic algorithms: Part I fundamentals. *University Comp* **15** (1993), pp. 170–181.
19. Alander JT. On optimal population size of genetic algorithms, Proceedings of CompEuro, Vol. 92. Silverspring, MD: IEEE Computer Society Press, 1992. p. 65–70.
20. J.E. Beasley, Lagrangean heuristics for location problems. *European Journal of Operational Research* **65** (1993), pp. 383–399.
21. R. Swain, A parametric decomposition algorithm for the solution of uncapacitated location problems. *Management Science* **21** (1974), pp. 189–198.
22. D. Serra and C. ReVelle, Market capture by two competitors: the preemptive location problem. *Journal of Regional Science* **3** (1994), pp. 549–561.