Annals of Operations Research 6(1986)35–46

DISCRETE SPACE LOCATION-ALLOCATION SOLUTIONS FROM GENETIC ALGORITHMS

C.M. HOSAGE* and M.F. GOODCHILD

Department of Geography, The University of Western Ontario, London, Ontario, Canada N6A 5C2

Abstract

Genetic algorithms are adaptive sampling strategies based on information processing models from population genetics. Because they are able to sample a population broadly, they have the potential to out-perform existing heuristics for certain difficult classes of location problems. This paper describes reproductive plans in the context of adaptive optimization methods, and details the three genetic operators which are the core of the reproductive design. An algorithm is presented to illustrate applications to discrete-space location problems, particularly the *p*-median. The algorithm is unlikely to compete in terms of efficiency with existing *p*-median heuristics. However, it is highly general and can be fine-tuned to maximize computational efficiency for any specific problem class.

Keywords and phrases

Genetic algorithm, adaptive search, heuristic, discrete-space.

1. Introduction

The purpose of this paper is to examine the possibility of using a class of adaptive search strategies known as genetic algorithms for solving location problems. Genetic algorithms are based on an analogy between optimization and the evolutionary behaviour of biological systems, which is exploited to produce a number of operators designed to move a set of solutions toward a global optimum. The basis of the analogy is as follows. The genetic code which defines each individual also defines that individual's ability to survive and breed, and thus propagate the code to succeeding generations. The probability of survival into the next generation in a given environment can therefore be thought of as a function of the genetic code, and the objective of the species as a search for the optimum code for that environment. Modification of the code occurs during sexual reproduction, when the offspring acquire sections of each parent's code, and in mutation.

*Present address: Digital Equipment Corporation, MK02-2/K03, Merrimack, NH 03054, USA.

The genetic analogy is established by a correspondence between the solution space of a location problem and a genetic code, in the form of a one-to-one mapping between each point in the solution space and each possible code. An initial population is established by random selection of locations and corresponding codes. The probability of a solution being involved in breeding the next generation, or the next cycle of the algorithm, is made to depend on the value of the objective function at that point in the solution space. It can be shown that this process has a finite probability of reaching every point in the solution space, and that the population will migrate toward the global optimum in successive generations. In particular, studies have shown that as adaptive sampling strategies, genetic algorithms are robust in multimodal and high-dimensional search spaces, suggesting that they might perform well on certain standard location problems. Bethke [1] has claimed that in general they are more likely to find the true optimum than hill-climbing strategies and are less expensive than enumerative search.

Several recent papers ([2,8,7]) have explored the application of a conceptually similar technique to the travelling salesman problem. Simulated annealing exploits an analogy between optimization and the random evolution of a physical system, as expressed in statistical mechanics. In a similar way, the probability of acceptance of a particular solution is made to depend on its associated objective function value: however, this appears to be the extent of the analogy as there is no equivalent of the concept of a biological population.

Following this very general introduction, the next section will discuss genetic algorithms in greater detail. This will be followed by a review of possible areas of application in location theory, and by a detailed study of the p-median problem.

2. Genetic algorithms

The object of genetic algorithms is to extract the mechanisms of adaption from natural systems and apply them to artificial systems. Three essential features of an adaptive plan are:

- E the environment of the system undergoing adaption
- τ the adaptive plan which determines successive structural modifications in response to the environment
- μ a measure of the performance of different structures in the environment.

This formalism also adopts a discrete time scale, $t = 1, 2, 3 \ldots$. These instants in time can be interpreted in various ways in different applications.

The operation of the adaptive genetic plan is as follows. Each individual within E is represented by some alphabet, for example the binary alphabet $[0, 1]$. Strings of members of the alphabet are the genotypes or chronosomes, where the specific position within the string has meaning. The set of structures within E is the set A which is

the domain of action for the adaptive plan, and is the set of all strings of length s that can be formed from the alphabet. A member of A is a chromosome with the form

$$A_i = (a_{1i}, a_{2i}, \ldots, a_{si}),$$

where the a's are the individual genes or alleles. A is a potential set, rather than a set which is actually rostered.

The adaptive plan will start with a random sampling or subset of A, where each sample has a value which is the fitness of the corresponding genotype. The adaptive plan τ observes the fitness values of the individuals and uses this information to produce a sequence of structures which replace extant structures in the subset considered, by making successive applications of a set of operators to be described below. Information from the environment aids in the selection of the fittest chromosomes or structures. Let the structure tried at time t be $A(t)$. Then the particular environment E confronting the adaptive system reacts by producing a signal $I_i(t)$, which is a performance evaluation of the structure tried,

$$I_i(t) = \mu_E(A_i(t)).$$

At any point in discrete time, the adaptive plan maintains a population of strings which is evaluated for performance. The new generation $A(t + 1)$ is produced from the better performers by calculating probabilities of selection which depend on the performance, as follows:

$$P(A_i(t)) = \frac{\mu_E(A_i(t))}{\sum_{A_j(t) \in A(t)} \mu_E(A_j(t))}.$$

Note that this selection criterion would be appropriate for maximization: to find minima, some suitable inversion would be applied.

The search is terminated when no significant improvement is obtained from the new generation. In order to ensure against a prolonged search, an adaptive system must exploit possibilities for improved performance during search. The manner in which this is accomplished becomes critical when the set of possible structures A is very large and the performance function has many local extrema. Exploitation of all possible structures is necessary to avoid entrapment in a local extremum, since an unexploited structure may be crucial to an optimal performance. The most efficient method of exploiting all possible structures is to test persistently and to incorporate structural properties associated with better performance.

Genetic algorithms perform well because of a property called "implicit parallelism", which identifies and preserves common components between strings having better than average performance. Recall that each point in the search space is represented by a string of length s. A schema is defined as a string of length s in which some positions are filled by specific members of the alphabet and the remaining positions are unfilled. For example, suppose $s = 3$ and the alphabet is binary and we wish to define a schema with a 1 in the first position. Then four chromosomes in A contain the schema as follows: (100), (101), (110) and (111).

The order of a schema h is the number of specified positions and two schemata are said to be competing if they have the same set of positions specified. Each schema corresponds to a region in the search space, which may or may not be singly bounded depending on the mapping between the search space and the string. Because the genetic algorithm reproduces strings in the population in proportion to their performance, Holland [5] has shown that the better performing of two competing schemata will be more prevalent in the next generation. The net result is a geometric growth or decline for each schema and the concentration of search in regions of good performance. Reproduction according to the fitness of schemata is the optimizing force in genetic algorithms.

Genetic operators are added to the mechanism to expand the search space without disrupting local searches in good regions. The three genetic operators are crossover, inversion and mutation. Crossover is a process analogous to sexual reproduction which recombines alleles by exchanging segments between pairs of chromosomes. A break is made at a randomly chosen point and two new strings are created by matching the portion of one string before the break with the portion of the other string after the break. For example, consider the two strings

01000011001100
10101101001010 .

A random crossover point is generated and the strings are broken:

01000 – 011001100
10101 – 101001010 .

Then the segments are exchanged to create two new strings:

01000101001010
10101011001100 .

It is easy to see that if both strings contain the same schema b, then both of the new strings will contain b.

By itself, the crossover operator is capable of diffusing through A to reach schemata that are not represented in the current population. This process constantly generates new schemata for evaluation, while testing extant schemata in new contexts. However, crossover alone would not lead to improved solutions without the pressure exerted by the reproductive mechanism. The simultaneous involvement of all schemata and the pressure of reproduction result in a diffusion outward from the schemata currently represented in $A(t)$ to schemata in areas of above-average performance.

> "Schemata of above-average performance can be produced from each other by relatively few crossovers . . . allowing thorough exploration of local optima in parallel fashion. All observed regions of high performance are exploited without significantly slowing the overall search for better optima" (Holland [5], p. 104).

The result is that older schemata associated with local optima steadily lose on the performance scale as better optima are located.

The operator of inversion produces a genetic modification within a single structure by inverting a randomly selected segment within a string. Two random integers in the range 0 to s are used to define the segment to be inverted. For example, the string

1011011110101

together with the random integers 3 and 10 would produce the new string

101 – 0111101 – 101 .

When the inversion operator is used in conjunction with crossover it will tend to replace an above-average schema with one of shorter length ([5], p. 106–7).

The genetic operator of mutation reintroduces alleles that have been eliminated during the deletion phases of the previous operators. The positions to be mutated are determined by a random process where each position has a small probability of selection, and a new structure is formed by replacing the values in the identified positions by a random value from the alphabet. The value of mutation is that it allows random alleles to be generated without respect to their associated performance. Alleles which occur in structures of below-average performance are especially vulnerable to elimination, and yet may be required for the adaptive plan to escape a local optimum.

3. Applications to location problems

A search strategy where implicit parallelism generates and uses information about a large number of schemata seems a reasonable approach to intractable problems. DeJong ([4], p. 25) has characterized genetic algorithms as "domain-independent generate and test heuristics" which are particularly suitable for solving NP-hard problems. Any domain specificity which may be found in a particular application can be attributed to the design of the evaluator and the representation of the space to be searched.

However, the use of genetic algorithms or reproductive plans in solving a particular problem may be inappropriate in spite of their theoretical advantages. Since the schemata function as building blocks which are recombined by crossover and inversion, the representation of a particular problem space must attribute no meaning to the relative positions of genes on the mating strings, and although different alphabets may be used, the algorithms operate most efficiently on binary strings. Thus, problem representation is crucial to the successful use of genetic algorithms, and requires particular thought in network and location problems.

Kainen [6] proposed the use of some aspects of the genetic analogy to generate shortest paths in networks. Weights would be assigned to each arc of the network incident at each node, and these could be used to guide an entity exploring the network. The entity would make decisions at each branching node based on probabilities computed from the arc weights. Kainen suggested that the arc weights could be based on shortest path trees from the origin and destination nodes, by counting the number of times each arc was involved in the shortest paths embedded in the two trees. Note, however, that both trees would contain the required solution; the procedure could only be regarded as a useful approach to solving the shortest path problem if branching probabilities could be based on locally derived weights which did not assume prior solution, and although the method uses some of the concepts discussed above, it can hardly be regarded as a direct application of genetic algorithms to a location problem.

Two studies have investigated the use of genetic algortihms for solving the travelling salesman problem. Wetzel [9] suggested three ways in which the solution space for an N-point tour could be mapped into a string of fixed length with associated alphabet. First, a string of length N could be used with an N-character alphabet. This would be highly redundant since only a very small fraction of such strings would represent legal tours, in which all of the integers from 1 to N occur exactly once. The second option would remove much of this redundancy, as follows. Without loss of generality, begin and end the tour at node N. All remaining nodes can be renumbered from 0 to $N-2$. Consider the numerical value of the binary string used to represent the entire tour. The index of the first point on the tour can now be obtained by finding the remainder after dividing by $N-1$. The index of the second point is obtained by truncating the quotient to an integer and dividing by $N-2$, after renumbering

the remaining nodes from 0 to $N-3$, and so on. In this way, the entire tour can be represented by a binary string of length equal to $\lceil \log_2 (N-1)! \rceil$. There is still a factor of 2 redundancy in the coding because each tour is represented in both forward and reverse directions. However, each binary string from 0 to $(N-1)!-1$ inclusive corresponds to exactly one legal tour. For example, the tour 6421356 maps to 0001000 and 6543216 to 1110111.

Wetzel's third suggestion was to represent each solution through its binary connectivity matrix. A tour through N nodes would require a string of N^2 bits, and only a very small fraction of strings would be legal tours. The probability that a legal string would result from a genetic operation would be extremely small.

Brockus [3] used the third of Wetzel's methods in implementing a genetic algorithm for the travelling salesman problem. Because of the problems with redundancy noted above, a modified approach was taken to the mating of two strings which has no direct genetic analogy. The two binary strings were first combined by a logical AND to produce two identical progeny. In general, the progeny would possess fewer 1's than either parent and would therefore represent incomplete tours. Each 1 in the first parent was then used to try to complete the tour of the first offspring, and similarly for the second parent and the second offspring. Finally, the remaining gaps in each tour were filled by brute force (specific details of this were not given).

Unfortunately, since Brockus used a modified crossover or mating procedure and does not report any inversion or mutation, there is some doubt whether the algorithm is capable of reaching all possible tours. Nevertheless, the reported comparisons of computational experience with other algorithms for the travelling salesman problem are encouraging.

Consider a facility location problem based on a network of N nodes. A binary string of length N could be used to represent the solution space, giving a 1 to each node which receives a facility. This would provide a one-to-one mapping of the solution space for the plant location problem, but would be less suitable for the p-median problem since only $\binom{N}{p}$ of the 2^N possible strings would be legal solutions with exactly p facilities open. On the other hand, a coding based on remainders similar to Wetzel's second method could be used to map the p-median space into strings of length $\lceil \log_2 N!/(N-p)! \rceil$. However, although almost all strings would correspond to valid nodes in the p-median solution space, each such node would map to $p!$ strings. There are consequently disadvantages to both methods for coding the p-median problem. In the implementation which follows, we have used the first alternative of adapting the plant location solution space with a penalty function. Further work will be needed to evaluate the second coding strategy for the p-median problem, and to investigate the first as a method of solving the plant location problem itself.

4. Implementation for the p-median problem

The objective function can be written:

$$\min Z = \sum_i \sum_j d_{ij} r_j \lambda_{ij},$$

where

r_j, $j = 1, n$	are the demands at the nodes
d_{ij}	is the distance from node i to node j
λ_{ij}	is a binary variable such that node j is assigned to a facility at node i when $\lambda_{ij} = 1$, else $\lambda_{ij} = 0$.

The constraints on the λ matrix require that each row sum to 1, that the trace of the matrix is equal to the number of central facilities p, and that no 1's be present in a row unless the diagonal element itself is a 1. However, since all nodes will be assigned to the nearest facility, the off-diagonal elements of λ follow once the diagonal elements are known. This allows us to map the solution space into a binary chromosome string of length N representing the diagonal elements. The weighted distance matrix with elements $r_j d_{ij}$ is used to compute the value of the objective function for any string.

With this problem representation, the genetic algorithms were implemented for the p-median problem. Specific considerations in the implementation were the design of the evaluator, the size of the reproducing population, the frequencies of application of the operators, and the number of generations to be produced. Initially, each generation was produced by complete replacement of the previous one, but later runs experimented with partial replacement. These two conditions will be referred to as non-overlapping and overlapping, respectively.

The initial form of the algorithm adhered closely to that described by Holland [5], and the effects of varying many of the parameters will be discussed below. New chromosomes were generated by crossover, inversion and mutation in each major cycle or generation. Each member of a breeding pair was selected independently from the current population using the probabilities discussed above, and a new pair of progeny generated by crossover at a randomly selected point. For a population of size M, M pairs were selected with replacement to give $2M$ progeny. One of each pair of progeny was then chosen randomly and rejected. The remaining M progeny were each subjected to inversion using two randomly chosen integers to define the section to be inverted. Finally, each of the new chromosomes was given a probability of 0.5 of being further subject to mutation. In this third operation, a single gene was chosen randomly from

each chromosome and replaced with a randomly chosen binary digit. The chance that a given chromosome chosen for mutation would actually be altered by the process is therefore 1/2, and the proportion of the entire generation affected by mutation is expected to be 1/4.

Many aspects of this implementation affect the efficiency and robustness of the algorithm, and there is clearly room for considerable fine-tuning. Many of the parameters have been subjected to experimentation during this study. However, excessive fine-tuning in the context of a specific problem such as the p-median would conflict with the advantage of the genetic procedure as a general approach to optimization, and might also constrain the ability of the algorithms to reach every part of the solution space.

The evaluator calculated the objective function Z, and then multiplied it by a penalty equal to the square of the absolute difference between the trace and the desired number of facilities p, plus 1:

$$Z' = Z(|\sum_i \lambda_{ij} - p| + 1)^2 .$$

The probability of being selected in the next generation was calculated by three different methods. Initially it was set proportional to $(1/Z')^c$, where c is some positive integer, since this would give the better performers selective advantage. However, varying c from 1 to 7 showed that it had little effect on the efficiency of the algorithm. The second method was to calculate the probability of selection based on the distance of Z' from some lower bound. In this case, the algorithm proved extremely sensitive to the value of the lower bound, which was set to some proportion of the least objective function value in the current population. The results of this second approach were an improvement over the first, but were highly erratic. It was reasoned that this was attributable to the variable advantages that were acquired by some individuals due to the range of values observed in the objective function. This was remedied in third approach by scaling all values of Z' to a fixed range in each generation. This approach proved most successful for the most difficult problems.

The number of generations the genetic algorithm is allowed to produce is critical in determining the nature of the final solution. Convergence is not a useful stopping rule because every generation including the last can be expected to produce far-from-optimal solutions, and since the search is simultaneously local and global, the algorithm may appear to converge whenever a local optimum is found. So the only reasonable stopping rule is a prior decision on the number of generations to be produced. Ideally, the number of generations would remain constant over many different sizes of problems. Computational experience indicated that more generations were required to solve problems with larger values of p, and that the number is also dependent on the type of generations used. Non-overlapping generations were more effective,

utilizing more chromosomes in the population, and less costly, requiring fewer generations. The number of generations required is also dependent on the size of the population (Bethke [1], p. 30), but since the size of the population was held at 20 plus or minus 5 throughout algorithm testing, no general conclusion can be made regarding the number required. One problem with 49 nodes and 9 medians required 700 generations to find the optimum solution.

Efficiency is clearly affected by the method used to generate the intial population and by the frequencies of application of the operators. Much better performance was obtained when the members of the intial population were constrained to the correct number of open facilities by generating N random numbers and opening facilities at the nodes which had been allocated the p largest numbers. At first, the inversion operator was applied to all surviving offspring, but this proved to be deleterious because inversion is disruptive and at such a high frequency it tended to produce a random search. It was subsequently applied to only 50% of the offspring. The mutation rate was initially held constant. However, since modification by mutation will always either add or delete an open facility and the p-median evaluator applied a penalty for an incorrect number of open facilities, this operator was likely to devalue an otherwise good chromosome. The operator was therefore modified to perform mutation when the chromosome had other than p open facilities, and to perform a swap by exchanging genes in two randomly chosen positions when exactly p facilities were open.

Failure of genetic algorithms occurs primarily because of genetic drift and misleading short schemata, neither of which would be corrected by a prolonged search. Genetic drift is the alteration of allele frequencies within the population due to stochastic effects associated with small populations. If a parent had fewer offspring than expected based on utility, and that number was low, it could easily have no descendants in succeeding generations. The mutation rate would have to approach that of a random search to counter genetic drift, but this would also destroy the good schemata found. Two methods to combat genetic drift have been devised that do not disturb the local searches. One ensures that the expected number of offspring are generated by limiting those parents that would be over-represented by chance in the next generation. The second method preserves the best string from the previous generation if it does not survive on its own (Bethke [1], p. 29). This method was used in the p-median application because the objective function values were often so close that the best string did not survive. Other possible approaches include the use of probabilities both to select pairs for breeding and to determine the chromosomes which will be replaced, and varying the relative frequencies of the three operators and the sequence in which they are applied.

The dramatic effects observed as a result of using certain selection probabilities and non-overlapping generations and not controlling the number of facilities in the initial chromosomes are due to genetic drift. Selective advantage and overlapping

generations limit the gene pool to just those genes in the best chromosomes, thus limiting the search. When the initial chromosomes are not forced to p open facilities, the algorithm must search for the correct chromosomes and incur severe losses in the gene pool.

The problem of misleading short schemata strikes at the basis of genetic algorithms, and is not easily resolved. The problem is that the best short definition schemata which do not contain the optimum may cause some alleles to become rare, thereby reducing the dimensionality of the search space. But in his theoretical analysis of the robustness of genetic algorithms, Holland [5] viewed the schemata of above-average performance as forming the basis of the technique's efficiency at finding the optimum.

Table 1

Performance of the genetic p-median algorithm

Length of string (N)	20	20	20	20
Number of facilities (p)	3	3	3	3
Population	25	25	25	25
Generations	120	150	180	210
Number of problems	100	100	100	100
Correct solutions found	69	85	84	89

Overlapping generations, selection based on $(1/Z')^2$.

Sample results of the implementation are shown in table 1. In all cases, the procedure was counted as having found the correct solution if that solution was present in the population at the last generation: since the best solution was always preserved into the next generation, there is no possibility of a best solution appearing and then later being lost. The correct solutions were found by using a branch and bound algorithm. All problems were generated by distributing nodes randomly in a square two-dimensional space and calculating straight line distances, and all used unit node weights. The figures suggest that the probability of finding the correct solution is still rising slowly in the last generation in each problem size.

5. Conclusions

An examination of the cases where the p-median genetic algorithm failed revealed that similarity between global and local optima in terms of the objective function was responsible for many problems. The algorithm is most likely to be trapped in a local optimum if the corresponding string is very dissimilar from the string of the true optimum, in other words if the mutually defining schema is very short. Un-

fortunately, this is very often the case in location problems, and its likelihood would seem to depend on the ordering of nodes in the basic coding scheme. Thus one would expect somewhat different performance if the coding order preserved spatial relationships than if it were essentially random. The definitions of both crossover and inversion operators ensure that the probability of two genes on a particular chromosome being preserved into the next generation is greater the closer together they are located on the string.

This paper is not intended as a detailed evaluation of the computational efficiency of genetic algorithms: to repeat a point made earlier, their advantages appear to lie in their generality in approaching difficult optimization problems rather than in the extent to which they can be fine-tuned to deal with a specific task. In principle, the same genetic approach could be used to manipulate binary strings for a large variety of location problems. The specific application would simply require one function to map each point in the solution space into a unique binary string, and a second to compute each string's evaluation function.

References

[1] A.D. Bethke, *Genetic Algorithms as Function Optimizers* (Logic of Computers Group, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1981).

[2] E. Bonomi and J.-L. Lutton, The *N*-city travelling salesman problem; Statistical mechanics and the metropolis algorithm, Ecole Polytechnique Fédérale de Lausanne, Département de Mathématique, Ecublens, Lausanne (1983).

[3] G.C. Brockus, Shortest path optimization using a genetics search technique, in: *Modeling and Simulation 14*, Proc. 14th Annual Pittsburgh Conf. (1983) p. 241.

[4] K. DeJong, in: *Synopsis: An Interdisciplinary Workshop in Adaptive Systems,* ed. J.R. Sampson (Logic of Computers Group, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1981) p. 26.

[5] J.H. Holland, *Adaption in Natural and Artificial Systems* (The University of Michigan Press, Ann Arbor, 1975).

[6] P. Kainen, in: *Synopsis: An Interdisciplinary Workshop in Adaptive Systems,* ed. J.R. Sampson (Logic of Computers Group, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1981) p. 75.

[7] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by simulated annealing, Science 220, 4598(1983)671.

[8] C.C. Skiscim and B.L. Golden, Optimization by simulated annealing: A preliminary computational study for the TSP. Paper presented at the NIHE Summer School on Combinatorial Optimization, Dublin, Ireland (1983).

[9] A. Wetzel, in: *Synopsis: An Interdisciplinary Workshop in Adaptive Systems,* ed. J.R. Sampson (Logic of Computers Group, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1981) p. 85.