ELSEVIER

European Journal of Operational Research 141 (2002) 480–494

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Solving large-scale maximum expected covering location problems by genetic algorithms: A comparative study

Haldun Aytug [a,*], Cem Saydam [b]

[a] *Department of Decision and Information Sciences, Warrington College of Business Administration, University of Florida, P.O. Box 117169, Gainesville, FL 32611-7169, USA*

[b] *Department of Business Information Systems and Operations Management, The University of North Carolina at Charlotte, Charlotte, NC 28223-0001, USA*

Received 29 November 1999; accepted 20 June 2001

Abstract

This paper compares the performance of genetic algorithms (GAs) on large-scale maximum expected coverage problems to other heuristic approaches. We focus our attention on a particular formulation with a nonlinear objective function to be optimized over a convex set. The solutions obtained by the best genetic algorithm are compared to Daskin's heuristic and the optimal or best solutions obtained by solving the corresponding integer linear programming (ILP) problems. We show that at least one of the GAs yields optimal or near-optimal solutions in a reasonable amount of time. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Genetic algorithms; Location; Integer programming

1. Introduction

Facility location/allocation represent a large and important class of problems that are highly visible both in the optimization literature and in the public sector [24]. The application areas include location/relocation of ambulances, siting of undesirable facilities, and retail outlet siting decisions [15,32]. Problems that focus on emergency response services – fire and ambulance – are one particularly critical example within this class of problems. Typically, they are large-scale optimization models that are difficult to solve in a reasonable amount of time, thus they require efficient heuristic solutions. The success of genetic algorithms (GAs) in solving complex combinatorial problems makes them strong candidates for solving this class of problems.

Various models have been developed to address these issues [26,27]. The first model, set covering location problem (SCLP), was developed by Toregas et al. [33]. SCLP searches for the least number of facilities required to cover a set of demand points within a given response-time standard. The next significant

[*] Corresponding author. Tel.: +1-352-392-2468; fax: +1-352-392-5438.
E-mail address: aytugh@notes.cba.ufl.edu (H. Aytug).

PII: S0377-2217(01)00260-0

milestone was the development of maximal covering location problem (MCLP) by Church and ReVelle [9]. Their approach accounted for limited resources by fixing the number of facilities and introduced the concept of maximizing the population covered within the response-time standard. Hogan and ReVelle [18] further extended the classic covering models (SCLP, MCLP) for multiple and backup coverage with and without the requirement of covering all demand points at least once. Lately, Gendreau et al. [16] developed a new double coverage model and a tabu search heuristic that provides near-optimal solutions in modest computing times.

In early 1980s Daskin introduced the maximum expected coverage location problem (MEXCLP) [12,13] that addressed the server congestion with a probabilistic optimization model. His approach attempted to maximize expected coverage given that the servers are busy and unavailable with a calculable system wide probability p. Daskin developed a single node substitution heuristic which he tested on a 55 node problem. In their 1988 paper Daskin et al. [14] integrated multiple, backup coverage models with the expected coverage models. Later, ReVelle and Hogan [28] developed the maximum availability location problem (MALP) which distributed a fixed number of servers to maximize the population covered with a server available within the response-time standard with reliability. They presented two versions of MALP, one with a system wide busy probability which is somewhat similar to MEXCLP, and the other version computed the local busy fractions for servers assuming that the immediate area of interest is isolated from the rest of the region. About the same time, Batta et al. [5] suggested "adjustments" to the MEXCLP to improve the accuracy of the expected coverage predicted by it. They proposed a two step heuristic that utilizes Larson's [21] hypercube optimization procedure. Saydam et al. [30] compared the accuracy of the predicted expected coverage of adjusted MEXCLP and found that MEXCLP provides "optimal" or "near-optimal" sets of locations, but, that there can be a significant over- or underestimation of coverage. Recently, Marianov and ReVelle [22] extended the MALP by utilizing queuing theory hence explicitly recognizing that the server busy probabilities are dependent on each other at the neighborhood level. This novel approach (Q-MALP) undoubtedly improves the overall realism of modeling.

GAs, first studied by Holland [19], are general-purpose search algorithms analogous to natural selection's survival of the fittest [17,23]. Artificial strings corresponding to chromosomes represent population members. The search starts with an initial population of strings. At each iteration individual strings are evaluated with respect to a performance criterion and, in return, assigned a fitness value, or strength. Based on their fitness values, strings are selected to construct the next generation by applying the genetic operators; selection, crossover and mutation. First two operators require two parents to be probabilistically selected based on their fitness values, where a high fitness value yields a higher chance of being selected for the next generation. After the parents are selected they may be carried to the next generation without altering (selection), or they may be crossed-over to generate the next set of solutions. Crossover operators range from a simple one-point crossover [17] to complex algorithms [34]. Mutation is a completely random operator, which is used to randomly alter the value of the digit of a selected string.

GAs have been used for many domains [17,23] including production scheduling [2,34], database query optimization [8], machine learning [4] and non-unicost set covering problems [7]. GAs are shown (empirically) to be very effective for solving similar problems optimally or near-optimally [1,6]. Given that there are no known polynomial time algorithms for the problems discussed above and inspired by Beasley and Chu's [7] successful application of a GA-based heuristic to non-unicost SCLP, a promising application of GAs is the MEXCLP in facility location/allocation.

The importance of the ambulance location/relocation problem is evidenced by recent debates that have appeared in the literature [5,22,27,30,32]. However, to date, there still is not a solution method that can address large-scale real-life problems in a reasonable time frame. A successful application of GAs to this problem would be a significant contribution to both the facility location/allocation field and the GA field. The successful application of such models can positively impact citizens in a given region in a number of ways. First and foremost is the increase in the number of lives saved by faster response times for optimally located ambulances. Second, the more cost efficient use of resources can improve the morale and performance of the ambulance personnel and minimize chronic high turnover rates that are very common in this profession.

The objectives of this research span two main areas (1) to solve large-scale MEXCLP quickly, (2) to develop novel coding schemes for GAs to work efficiently in this domain, and (3) to compare the solution quality and CPU time trade-offs of various approaches including Daskin's heuristic and the GA approach. The rest of the paper is organized as follows. In Section 2 we briefly restate the MEXCLP followed by Section 3. We then discuss different coding schemes and provide results of our computational experiments in Section 4. Section 5 summarizes our findings and includes recommendations for future studies. Sample crossover operators and a summary of the notation are given in Appendix A.

2. Formulation of MEXCLP

Daskin made a major contribution when he formulated the MEXCLP [12,13]. Briefly, MEXCLP can be stated as: Locate M facilities at possible sites on a network to maximize the population expected to be covered within a given service distance r and given the probability of each vehicle being busy. Critical to Daskin's formulation is the fact that if m units must cover a point geographically, and if each unit is busy with probability p, then the probability that the point is covered by at least one unit is $1 - p^m$. Daskin maximized the expected coverage as follows:

$$\text{Maximize} \quad \sum_{j=1}^{n} \sum_{i=1}^{M} (1-p)p^{i-1} h_j y_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{M} y_{ij} - \sum_{i=1}^{n} a_{ij} x_i \leqslant 0 \quad \forall j, \tag{2}$$

$$\sum_{i=1}^{n} x_i \leqslant M, \tag{3}$$

$$x_i = 0, 1, 2, \ldots, M \quad \forall i, \tag{4}$$

$$y_{ij} = 0, 1 \quad \forall i, j, \tag{5}$$

where M is the maximum number of facilities to be located and n is the number of nodes;

x_i = number of facilities located at node i,

$$y_{ij} = \begin{cases} 1 & \text{if node } j \text{ is covered by at least } i \text{ facilities,} \\ 0 & \text{if node } j \text{ is covered by less than } i \text{ facilities,} \end{cases}$$

h_j = the demand generated at node j.

The objective function (1) maximizes the expected number of demands that can be covered, constraint (2) computes the number of times node j is covered and relates the decision variables y_{ij} to the first set of decision variables, x_i. Constraint (3) specifies the maximum number of facilities to be located on the network. Constraint (4) allows multiple units to be located at any node. We will refer to this formulation as the integer linear programming (ILP) formulation.

It is important to note that the MEXCLP can be reformulated using fewer variables using a nonlinear objective function [31]. Let y_j denote the number of times node j is covered. Given n demand nodes in the network, we can represent the expected coverage for each of the demand nodes by $h_j(1 - p^{y_j})$ for all j. We know that the number of facilities capable of covering node j is given by $\sum_{i=1}^{n} a_{ij}x_i$. Therefore, we let $\sum_{i=1}^{n} a_{ij}x_i = y_j$ for all j. The nonlinear version of the MEXCLP can be formulated as follows:

$$\text{Maximize} \quad \sum_{j=1}^{n} h_j(1 - p^{y_j}) \tag{6}$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_{ij}x_i = y_j \quad \forall j, \tag{7}$$

$$\sum_{i=1}^{n} x_i \leqslant M, \tag{8}$$

$$x_i = 0, 1, 2, \ldots, M \quad \forall i, \tag{9}$$

$$y_j = 0, 1, 2, \ldots, M \quad \forall j. \tag{10}$$

Although the two formulations are theoretically identical, the nonlinear formulation of MEXCLP lends itself for easier implementation via genetic algorithms. The objective function (6) can be directly coded as the fitness function. Given a feasible solution, it is rather simple and efficient to count the number of times each node is covered with a conditional vector multiplication.

3. Research methodology

As stated earlier, we are particularly interested in solving large-scale applications of the maximum expected coverage model applied to ambulance location/relocation problems. A frequently used service goal in emergency medical service systems (EMS) is to respond to (or *cover*) a certain percentage of demands within a fixed time limit. This service goal originated in a 1973 federal EMS funding program as responding to 95% of demands within 10 minutes (for urban areas). The percentage of demands answered within some predetermined response time threshold is commonly referred to as the system's *coverage*. The size of the ambulance fleet, the locations of the vehicles, and the unit busy probabilities are factors that affect the EMS system's ability to improve its coverage.

Theoretically, the maximum expected coverage model should prescribe optimal locations for units and accurately predict the region wide expected coverage. However, it is a fact that these models can over- or underestimate coverage prediction by up to 25% in extreme operating environments (heavy call traffic) [5,30]. One of the most significant factors that affects the accuracy of the maximum expected covering model results is the degree of demand data aggregation which determines the size of the coverage matrix used to represent the region. An obvious remedy is to increase the matrix size with minimal aggregation but the increase in the size of this matrix increases the solution time exponentially since the problem is NP-complete. GAs' success on solving many combinatorial problems efficiently makes them likely candidates for solving this problem.

GAs work very well on unconstrained optimization problems without any further engineering on its search operators. However, when the search space is constrained the regular search operators do not guarantee feasible solutions. In such circumstances the genetic operators need to be engineered to guarantee feasible solutions [10] or penalties are imposed on infeasible solutions to make their survival less likely [1,23,29]. Using penalty functions though does not guarantee a feasible solution when the algorithm "converges". Since the maximum expected coverage problem is constrained, it is crucial to identify a coding scheme that guarantees feasible solutions.

Houck et al. [20] discuss the use of GAs for the location–allocation problem and report its success when compared to existing heuristic procedures. Even though the authors do not report exactly which genetic operators dominate the GA search, the superior performance of the GA is quite encouraging. They use real-number encoded GA strings with several different crossover and mutation operators, none of which are adaptable for the MEXCLP. However, using random keys for some constrained optimization problems has yielded promising results [6]. The resource allocation problem (RAP) which is a simplified version of facility location/allocation problem has been solved effectively by GAs using random keys [6], an idea investigated by Bean [6]. In this representation, each bit is encoded with a random number (encoding). The genetic operator and function evaluator decodes it in a way to guarantee feasibility. In this research, we also investigate the effectiveness of random keys for the MEXCLP.

The research methodology we followed has two stages. First, efficient coding schemes for solution representation are identified, coded and tested on sample problems. Second, the effectiveness of the proposed approach is determined by comparing the quality of solutions generated by the GA approaches to Daskin's heuristic detailed in [13], and the solutions obtained using the traditional ILP approach [25,30].

3.1. Design of the genetic algorithm

There are five key issues in designing a GA algorithm. (1) Selecting an appropriate representation, (2) an effective crossover operator, (3) an effective mutation operator, (4) a feasible initialization and (5) appropriate crossover and mutation rates that will create the best answer when the algorithm stops. In the next subsections we explain the alternatives we have considered.

3.1.1. Representation schemes

We employed the following three representation schemes. First representation uses random-key coding [6]. A solution is represented by a random key coded string s of length n^*K where s_i is randomly assigned a value between 0 and 1. The number of ambulances in a given node is represented by K random numbers. K is selected such that, 2^K is the smallest integer greater than the number of ambulances allowed at one node. For example, when K is equal to 2 we are restricting the solutions to at most 3 $(2^K - 1\ )$ ambulances on any node. To eliminate the potential that a parent with high random numbers can eliminate the solution from a parent with very low random numbers each string is normalized to satisfy the following constraint:

$$\sum_i 2^{K-1-(i \bmod K)} s_i = M. \tag{11}$$

Eq. (11) is equivalent to Eqs. (3) and (8).

Given $n = 4$, $K = 2$ and $M = 3$. A feasible string s may look like

0.80.1 | 0.10.2 | 0.10.1 | 0.10.4.

The decoding process requires ranking the bit indices in descending order based on the corresponding random number value. Starting with the bit that has the highest random number ambulances are assigned (i.e., the corresponding bit location is set to 1) to the corresponding bit location. An assignment to bit i corresponds to $2^{K-1-(i \bmod K)}$ ambulances at that location. All bit locations that are not assigned an ambulance are set to 0. For the string above, ranking the bits based on the random numbers creates the following order of bit indexes (0,7,3,1,4,5,6). Bit 0 is set to 1 which is equivalent to assigning two ambulances to node 0. Setting bit 7 to 1 is equivalent to assigning one ambulance to node 3. Since we have assigned all ambulances bits 3,1,4,5 and 6 are set to 0 yielding the binary representation

10 | 00 | 00 | 01.

Second representation is an integer representation. A solution s has n bits, each corresponding to the number of ambulances in that location i.e., $s_i \leqslant M, \sum_i s_i = M$. This is used in Crossover four. Third representation is also an integer string such that a solution s is a string of length M (index-based representation). Each bit in s corresponds to the node on which an ambulance is placed, i.e., $s_i \in \{1, \ldots, n\}$, $i = 1, \ldots, M$. Crossover five uses this representation.

3.1.2. Crossover operators

Since each crossover operator assumes a particular representation not all will work with each representation. Crossover operators one, two and three work with random key representation. Crossover four and five uses integer and index-based representations respectively. The third representation is only used by Crossover five.

Crossover one (one-point crossover) involves selecting a random crossover sight on the coded parents and swapping the sub-strings of both parents. Once the children are obtained they are normalized so that Eq. (1) holds.

Crossover two (row crossover) conceptualizes the nodes as a matrix and performs uniform crossover by row. The first row of one parent is crossed with the first row of the other parent and so on. Crossover three (column crossover) works exactly like Crossover two but uses columns instead of rows.

These three operators use random-key coding. Since the issue of infeasibility is resolved by the random-key representation, crossover operators one through three do not need to consider infeasibilities in creating the children.

Crossover four (best coverage crossover) identifies the ambulance locations with the highest coverage from both parents. Top M locations (regardless of from which parent it is received) are assigned to one child (strong child) and the remaining M are assigned to the second (weak child).

The implementation of Crossover five (ambulance swap) requires that the crossover location be identified not in terms of nodes but in terms of ambulances (index based). This crossover operator creates a random number u between 1 and $M - 1$ and swaps u leftmost ambulances of each solution (see Appendix A for examples of each operator).

3.1.3. Mutation operator

The mutation operator is used to "jump-start" the current solution so as to escape a possible local optimum and find a new neighborhood with a potentially more promising solution. It is a random operator and there are many variants of it in literature depending on the representation used. Our implementation alters each bit location randomly if the random key is used. For integer implementation the effect of changing one bit is countered by changing another bit so as to preserve Eq. (11).

3.1.4. Initialization

We developed two different initialization routines for the initial populations: random and marginal coverage heuristic. Random initialization requires creating feasible solutions randomly. Random key coding requires no feasibility check since the decoding algorithm guarantees feasibility. When using integer coding though M ambulances were assigned to at most M random locations (with the possibility of assigning more than one ambulance in one location) to guarantee feasibility.

Marginal coverage heuristic requires identifying locations that can offer the highest marginal objective value. Those locations are ranked and M ambulances are assigned to at most M locations with the highest marginal objective value. Note that, multiple ambulances can be assigned to the same location as long as the marginal value of assigning the next is higher than assigning that ambulance somewhere else. Since this method can create only one such solution and not a population, after two copies of this solution the rest of the population members are created randomly.

3.2. Choosing the GA parameters

Any GA implementation requires setting mutation, crossover and population size parameters. Aytug and Koehler [3] suggest that in the worst case a large population size and a high mutation rate regardless of the crossover rate reduces the number of iterations required before the optimal is seen for a given probability (i.e., the probability that the optimal has been seen after t iterations). Our sample runs support that this is true for deceptive functions, however on polynomial functions (with potential multiple local optima) there is a trade-off between mutation and crossover rates but large population sizes are favorable (at the cost of computation time) in either case. Since there is no formula that would tell us the appropriate value for mutation and crossover we conducted some preliminary experiments varying the mutation and crossover rates for each crossover operator and initialization schemes. Crossover rates around 0.7 and mutation rates around 0.03 yielded the best results regardless of the crossover and initialization procedures used. As a result we ran the remaining experiments using a crossover rate of 0.7 and a mutation rate of 0.03 regardless of other parameters. The population size N is given by $N = \max(100, 0.75n)$.

3.3. Experimental setup

The MEXCLP is fully defined by the number of nodes, the distance matrix, the coverage radius of servers, the number of servers, the distribution of calls among the nodes and the system wide probability of a server being busy. To evaluate the effectiveness of the GA approach we generated test problems by varying the number of nodes which is thought to be the most significant factor negatively influencing the solution time, the number of servers and the coverage radius.

We assumed that each node represents a square mile. For each level of number of nodes, we randomly distributed 10,000 calls to the geometric center of each node. We then calculated coverage matrices using Euclidean distances. Finally, we calculated the system wide busy probability of a server by using the Erlang's loss equation ($\rho = \lambda/M\mu$), where M is the number of servers and λ/μ is arbitrarily set to 2.0 for the first two sets of experiments. This resulted in $\rho = 0.40$ when $M = 5$, and $\rho = 0.22$ for $M = 9$, whereas λ/μ is set to 4.0 for the third set of experiments which resulted in $\rho = 0.40$ when $M = 10$, and $\rho = 0.10$ for $M = 20$. These settings are consistent with the literature [5,30].

We conducted three sets of experiments. The first limited set was conducted on 48 problems to determine if there is a dominant configuration among the initialization and crossover combinations. The second set consisted of 140 problems to compare the performance of the best GA obtained in Experiment one to random search so as to determine a lower bound for these problems. The last experiment was conducted on problems we considered hard for optimization (i.e., prohibitive computational times) and wanted to compare the best GA computation time and solution quality. We employed CPLEX as the solver to obtain the optimal solutions to the linear integer formulation as defined by Eqs. (1)–(5). In cases when CPLEX failed to return the optimal solution after a minimum of about 12 hours, we terminated the program and reported the best-known integer solution. These results were then used to benchmark the best GA solutions and solutions from Daskin's [13] heuristic. The problem parameters used for each experiment are given in Table 1. All algorithms are coded in C++, ILP formulations are solved using CPLEX 4.0.7 [11], and the study is conducted on a PC with Pentium II processor rated at 350 MHz.

Table 1
Display of levels used to generate the test problems

Parameters	Experiment one	Experiment two	Experiment three
Number of nodes (n)	100, 225, 400	100, 225, 400, 625, 900, 1225, 1600	400, 625, 900, 1225, 1600
Number of servers (M)	5, 6, 7, 8	5, 6, 7, 8, 9	10, 12, 14, 16, 18, 20
Coverage radius (r)	2, 3, 4, 5 (miles)	2, 3, 4, 5 (miles)	6, 7, 8 (miles)

4. Experiments and results

We arbitrarily set the iteration limit to 100 to get a feel for the quality of the GA solutions with a rather small number of iterations. Consequently, the GA searches a minimum of 10,000 solutions (for $n = 100$) and a maximum of 30,000 solutions (for $n = 400$) some of which are duplicates. Note that the size of the solution space can be estimated by

$$\binom{n+M-1}{n-1}.$$

For example, when $n = 100$ and $M = 5$, the GA searches at most 0.0000181% of the search space.

To determine the best crossover and initialization combination we ran the GA for $n = 100, \ldots, 400$, $M = 5, \ldots, 8$, and $r = 2, \ldots, 5$. The quality of solutions generated by these experiments is shown in Table 2. The performance statistics (the GA solution divided by the optimal solution multiplied by 100) in Table 2 indicate that the combination of "Crossover five" and the "random" initialization provides excellent results. It is also important to note that the heuristic initialization always starts with a better solution but the GA with random initialization seems to make up for the difference in merely 100 iterations.

Fig. 1 displays the average CPU time behavior of each operator. Because of the coding used Crossover five scales nicely compared to others as the number of nodes in the problem increases. Crossover four is the slowest due to its additional step to calculate the marginal contribution of placing a server on a node.

Table 2
Summary statistics by crossover and initialization type as a percentage of the optimal solution (average, [min, max])

Crossover type	Initialization routine	
	Random	Heuristic
1	92.3, [82.6, 99.7]	92.3, [83.1, 99.7]
2	92.2, [83.1, 99.8]	92.0, [83.0, 99.8]
3	91.9, [82.3, 99.9]	92.3, [84.4, 99.7]
4	92.3, [83.2, 99.7]	90.4, [79.4, 100.0]
5	99.1, [95.8, 100.0]	99.1, [95.3, 100.0]

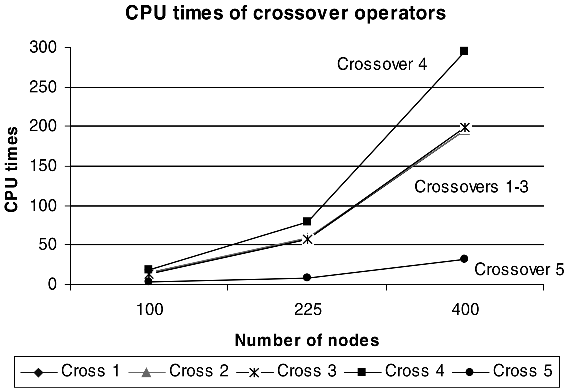

Fig. 1. Average CPU times for crossover operators 1–5.

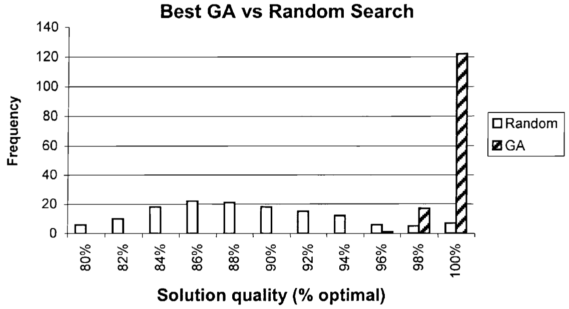

Fig. 2. Frequency distribution of the best GA and random solutions.

Crossovers one, two and three have virtually identical CPU times. Since Crossover five yields the highest quality solutions in the lowest amount of time, experiments two and three are conducted using this operator with random initialization.

We then compare the GA using Crossover five with random initialization to ILP solved by CPLEX both in run-time, scalability and solution quality. We also generate random solutions to demonstrate that the GA actually searches the solution space effectively. To demonstrate the effectiveness of our implementation versus randomly selected solutions, we solved 140 problems (Experiment two) using Crossover five and random initialization with 100 iterations and compared the results to randomly generated solutions. To account for the overhead of the GA, we generated and evaluated random solutions in the amount of time it took for the GA to stop after 100 iterations. Consequently, for each problem the number of randomly generated solutions is greater than the number of solutions generated by the GA. The distribution of the solutions generated by the GA with Crossover five with random initialization versus random search routine are shown in Fig. 2. Clearly, the distribution of the GA solutions is positively skewed leaving only about 3% of the solutions with an objective function at most 4% away from the optimal solution. Solutions obtained using random search were on average within 87% of the optimal with a range between 78 to nearly 100%. By contrast, the best GA solutions averaged 99.14% with the worst solution being off by less than 5% of the optimal solution.

Experiment two showed that the GA finds a very good solution very quickly but it stalls. To remedy this situation we modified the GA by adding a simple local (neighborhood) search algorithm at the end of 100 iterations. The local search (LS) algorithm is a greedy search algorithm that is designed to find a better solution quickly if there is one in the neighborhood. Even though it has a beam size parameter b, LS is different from a beam search. That is, a beam search algorithm will commit to b solutions only after it has inspected all solutions that can be inspected based on the current solutions in the beam. LS commits to a solution as soon as it finds a better solution. Since the algorithm only accepts improved solutions it will stop in finite steps. We first provide an algorithmic description of LS and then describe the main ideas.

Let

B be the set of best b GA solutions,
B_z be the z^{th} element in B, $z = 1, \dots, b$,
d be the radius of the neighborhood that has a server located at its center, and
N_j be the set of nodes in the neighborhood of node j within radius d.

Given B and d the algorithm will return the best solution it could find.

Algorithm LS.

```
While there is improvement
   $i=0$
   $\bar{f} = \max_{z=1..b}\{f(B_z)\}$
   While $i < b$
      $s = B_i$
      $j = 0$
      While $(j < M)$
         $x = s_j$
         $k = 0$
         While there are nodes in $N_j$
            $s_j = N_{jk}$ (insert a neighboring node into the current solution)
            If $f(s) > \min_{z=1..b}\{f(B_z)\}$ then
                 Insert $s$ into $B$
                 Delete $B_y$ from B such that $f(B_y) = \min_{z=1..b}\{f(B_z)\}$
            Endif
            Increment $k$
         End while $k$
         $B_{ij} = x$
         Increment $j$
      End while $j$
      Increment $i$
   End while $i$
   If $(\bar{f} = \max_{z=1..b}\{f(B_z)\})$ then (note that B might have changed)
      improvement = false
   Else
      $\bar{f} = \max_{z=1..b}\{f(B_z)\}$
   Endif
End while improvement
Output, $\bar{f} = \max_{z=1..b}\{f(B_z)\}, B_{z^*}$
```

The algorithm keeps a list of b solutions (B) and picks an ambulance location from a given solution $B_z \in B$. It moves the ambulance to each of the neighboring locations one at a time keeping everything else fixed. Any time a solution that is better than the b known solutions is found it replaces the worst of b solutions. The neighborhood search of B_z is completed when the neighborhood of each ambulance location in B_z is searched one at a time. When the neighborhood of each ambulance of B_z is searched the algorithm moves to B_{z+1}, and repeats the search. The algorithm scrolls through the list of b items in a circular fashion (i.e., after the first b moves the algorithm goes back to the beginning of the list) until it cannot improve the solutions any further.

Experiment three is designed to compare the GA, GALS, and Daskin's heuristic to solutions obtained by CPLEX on large-scale problems. Experiment three revealed that as the problem size increases (in terms of ILP formulation variables and constraints) both the GA and GALS start to be useful in terms of their time versus quality trade-off with respect to ILP and Daskin's heuristic. Table 3 displays the solution quality of each heuristic as a percentage of the best ILP solution obtained via CPLEX. (Entries in Table 3 that are greater than 100% correspond to the problems where either heuristic created better solutions than CPLEX.) Overall the GALS matches or exceeds (albeit slightly) the best integer solutions CPLEX found. In 14 cases the GALS found the optimal solution and in 27 other cases it created solutions that are better than

Table 3
Comparisons of heuristics

Nodes	GA average, [min, max]	GALS average, [min, max]	Daskin average, [min, max]
400	99.68, [99.25, 99.99]	99.98, [99.91, 100.00]	99.92, [99.81, 99.98]
625	99.24, [98.53, 99.87]	99.95, [99.81, 100.00]	99.96, [99.74, 100.30]
900	99.07, [97.42, 101.00]	100.24, [99.76, 102.56]	100.18, [99.58, 102.60]
1225	98.82, [96.94, 100.66]	100.45, [98.73, 102.78]	100.49, [99.29, 102.98]
1600	98.96, [95.85, 103.53]	101.08, [99.29, 104.93]	100.93, [98.52, 104.97]

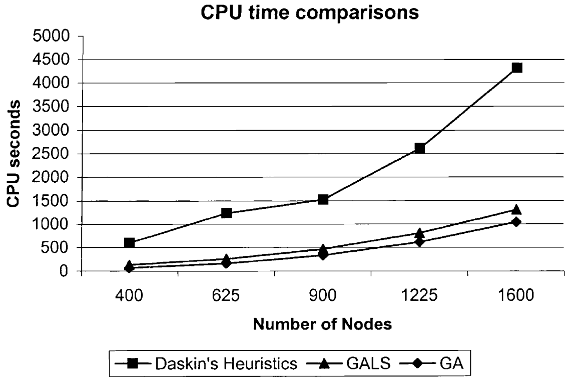

Fig. 3. Average CPU times for the GA, GALS and Daskin's heuristic.

the best integer solution found by CPLEX. Daskin's heuristic is also very good as it almost matches the quality of the GALS but at a high computational burden as shown in Fig. 3. For a given set of nodes the average computation time of GALS varied by about 40% of the mean as the number of ambulances changed (in the average, about 40% less for $M = 10$ and about 40% more for $M = 20$). A similar pattern is observed for Daskin's heuristic. The CPU times for the ILP are not reported since 44 out of 90 failed to stop and we stopped them after several hours. Thirty-eight of these cases are for problems with 900 or more nodes confirming that computational requirements of such problems are unpredictable and likely to increase exponentially.

5. Conclusions

We have examined four approaches for solving large-scale maximum expected coverage location problems: an integer programming approach, Daskin's heuristic, a GA, and a GA with LS. The integer programming approach theoretically produces optimal solutions but it may take an unacceptably long time. The GA approach that uses the random initialization and Crossover five yields high quality solutions in predictable times. Daskin's heuristic produces better quality solutions than the GA but is computationally burdensome. However, the GALS yields the best solutions with slightly worse computational demands than GA.

Crossover five (the ambulance swap) is the dominant operator for this problem. The high performance of this operator can be explained as follows. The objective function is additive, therefore when an ambulance is passed onto a new solution it preserves its marginal contribution. When a crossover is performed on parents with high fitness, it is likely that ambulances that were swapped were in the same neighborhood perhaps a couple of nodes off. Consequently, the resulting children will have similar structures preserving high performing subschemas from one generation to the other. Even though Bean [6] suggests very encouraging solutions for set covering problems, the random key encoding has not contributed much to the operation of the GA for this problem. The integer representation and Crossover five ensures feasibility, is fast and is very easy to implement.

The GALS is very robust. The analysis of the results based on Experiment three shows that the worst case solution is merely 1.27% of the optimal solution (Table 3). Furthermore, on average, in a fraction of the time it generates identical or better solutions than CPLEX. Table 3 suggests that the number of nodes in the problem does not affect the quality of GALS solutions. On the contrary, as the problem size increases, the likelihood of CPLEX failing to find the optimal solution increases. While the ILP is restricted to linear or linearizable objective functions, Daskin's heuristic is specific to the MEXCLP objective function (Eq. (1)), the GALS and the GA can be used for any objective function. Our results strongly suggest that the prescribed GALS approach is an effective strategy for solving large-scale expected coverage location problems.

Appendix A. Crossover operators

Consider a grid of nine nodes where we are trying to place two ambulances. The following two strings bp_1 and bp_2 are two feasible solutions using random-key coding and ip_1 and ip_2 are the corresponding integer solutions. Strings bc_1, bc_2, ic_1 and ic_2 correspond to the binary and integer children. Since each position corresponds to a binary value (0,1) we need strings of length 18 (2×9) where each bit pair corresponds to a node that can have up to three ambulances. The decoding algorithm of random-key coding ranks the bit positions and distributes the ambulances starting from top until all are distributed. The bits that are in bold below correspond to top M location. Note that

let

$$bp_1 = 0.1\ 0.1\,|\,0.05\ 0.3\,|\,\mathbf{0.4}\ 0.01\,|\,0.2\ 0.1\,|\,0.09\ 0.15\,|\,0.1\ 0.05\,|\,0.02\ 0.13\,|\,0.02\ 0.03\,|\,0.05\ 0.1,$$

$$bp_2 = 0.1\ \mathbf{0.19}\,|\,0.06\ 0.1\,|\,0.1\ 0.1\,|\,0.1\ \mathbf{0.3}\,|\,0.1\ 0.1\,|\,0.05\ 0.15\,|\,0.01\ 0.1\,|\,0.09\ 0.15\,|\,0.12\ 0.08,$$

then

$$ip_1 = 0\,|\,0\,|\,2\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0,$$

$$ip_2 = 1\,|\,0\,|\,0\,|\,1\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0.$$

A.1. Single-point crossover

This is the classical crossover operator published in the literature. It works with the random key coding. Given

$$bp_1 = 0.1\ 0.1\,|\,0.05\ 0.3\,|\,0.4\ 0\,||\,0.01\,|\,0.2\ 0.1\,|\,0.09\ 0.15\,|\,0.1\ 0.05\,|\,0.02\ 0.13\,|\,0.02\ 0.03\,|\,0.05\ 0.1,$$

$$bp_2 = 0.1\ 0.19\,|\,0.06\ 0.1\,|\,0.1\,||\,0.1\,|\,0.1\ 0.3\,|\,0.1\ 0.1\,|\,0.05\ 0.15\,|\,0.01\ 0.1\,|\,0.09\ 0.15\,|\,0.12\ 0.08.$$

Let the crossover site $k = 5$, $k = U(1..17)$. Then

$$bc_1 = 0.1\ 0.1\,|\,0.05\ 0.3\,|\,0.4\ 0.1\,|\,0.1\ 0.3\,|\,0.1\ 0.1\,|\,0.05\ 0.15\,|\,0.01\ 0.1\,|\,0.09\ 0.15\,|\,0.12\ 0.08,$$

$$bc_2 = 0.1\ 0.19\,|\,0.06\ 0.1\,|\,0.1\ 0.01\,|\,0.2\ 0.1\,|\,0.09\ 0.15\,|\,0.1\ 0.05\,|\,0.02\ 0.13\,|\,0.02\ 0.03\,|\,0.05\ 0.1.$$

After normalizing and decoding

$$ic_1 = 0\,|\,0\,|\,2\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0,$$

$$ic_2 = 0\,|\,0\,|\,0\,|\,2\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0.$$

A.2. Row (column) crossover

This crossover operator assumes that the problem can be described as an $n \times m$ matrix. We can conceptualize each solution as 3×3 matrix. Then

$$ip_1 = \begin{bmatrix} 0.1\ 0.1 & |0.05\ 0.3| & 0.4\ 0.01 \\ 0.1\ 0.1 & |0.09\ 0.15| & 0.1\ 0.05 \\ 0.02\ 0.13 & |0.02\ 0.03| & 0.05\ 0.1 \end{bmatrix}$$

and

$$ip_2 = \begin{bmatrix} 0.1\ 0.19 & |0.06\ 0.1| & 0.1\ 0.1 \\ 0.1\ 0.3 & |0.1\ 0.1| & 0.05\ 0.15 \\ 0.01\ 0.1 & |0.09\ 0.15| & 0.12\ 0.08 \end{bmatrix}.$$

Performing 3 single-point crossovers by using the ith ($i = 1..3$) row (column) from each parent yields the children. As in single-point crossover the children have to be decoded.

A.3. Best coverage crossover

Each node that has an ambulance is ranked by the marginal contribution of each ambulance to the objective function. Regardless of who the parent is those nodes are ranked (there should be total of $2M$ ranked indices) and the first M indices are given to the first child and the remaining to the second child. After initialization, this crossover is always feasible.

A.4. Representation

Consider the following marginal contribution of placing an ambulance in a node out of nine. Note that, based on our objective function adding the 2nd ambulance to a node has diminishing returns.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1st	0.2	0.01	0.3	0	0.15	0.75	0.2	0.4	0.12
2nd	**0.11**	0.001	**0.1**	**0**	0.03	0.3	0.05	0.12	0.06
3rd	...								

If the following are the parents:

$$ip_1 = 0\,|\,0\,|\,2\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0,$$

$$ip_2 = 1\,|\,0\,|\,0\,|\,1\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0,$$

the ranking of the indices are $ip1_{1,\mathbf{2}}$, $ip2_{1,\mathbf{0}}$, $ip1_{2,\mathbf{2}}$, $ip2_{1,\mathbf{4}}$.

Consequently the children are

$$ic_1 = 1\,|\,0\,|\,1\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0,$$

$$ic_2 = 0\,|\,0\,|\,1\,|\,1\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0.$$

A.5. Ambulance swap

First the number of ambulances to swap (k), $k = U(1..M-1)$, is selected randomly. The locations of the leftmost k ambulances are swapped. For the sake of this example assume we have three ambulances $(M = 3)$ and nine nodes $(n = 9)$. Given

$ip_1 = 3\,|\,3\,|\,5$ (there are two ambulances on node three and one on five),

$ip_2 = 1\,|\,4\,|\,6.$

Let $k = 1$. This is equivalent to taking one ambulance from node 3 of ip_1 and placing it on node 3 of ip_2 and taking the ambulance on node 1 of ip_2 and placing it on the node of ip_1.

$$ic_1 = 1\,|\,3\,|\,5,$$

$$ic_2 = 3\,|\,4\,|\,6.$$

Assume $k = 2$ rather than one. Then

$$ip_1 = 3\,|\,3\,|\,6,$$

$$ip_2 = 1\,|\,4\,|\,5.$$

References

[1] E.J. Anderson, M.C. Ferris, Genetic algorithms for combinatorial optimization: The assembly line balancing problem, ORSA Journal on Computing 6 (2) (1994) 161–173.

[2] H. Aytug, G.J. Koehler, J.L. Snowdon, Genetic learning of dynamic scheduling within a simulation environment, Computers & Operations Research 21 (8) (1994) 909–925.

[3] H. Aytug, G.J. Koehler, Stopping criteria for finite length genetic algorithms, ORSA Journal on Computing 8 (2) (1996) 183–191.

[4] H. Aytug, S. Bhattacharyya, G.J. Koehler, J.L. Snowdon, A review of machine learning in scheduling, IEEE Transactions on Engineering Management 41 (2) (1994) 165–171.

[5] R. Batta, J.M. Dolan, N.N. Krishnamurthy, The maximal expected covering location problem: revisited, Transportation Science 23 (1989) 277–287.

[6] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, INFORMS Journal on Computing 6 (2) (1994) 154–160.

[7] J.E. Beasley, P.C. Chu, A genetic algorithm for the set covering problem, European Journal of Operational Research 94 (1996) 392–404.

[8] K. Bennett, M.C. Ferris, Y.E. Ioannidis, A genetic algorithm for database query optimization, in: Proceedings of the Fourth International Conference on Genetic Algorithms, University of California, San Diego, July 13–16, 1991, pp. 400–407.

[9] R. Church, C. ReVelle, The maximal covering location problem, Papers of the Regional Science Association 32 (1974) 101–118.

[10] G.A. Cleveland, S.F. Smith, Using genetic algorithms to schedule flow shop releases, in: Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 160–169.

[11] CPLEX (1995) Using the CPLEX Callable Library, CPLEX Optimization, Inc., Incline Village, NV.

[12] M.S. Daskin, Application of an expected covering location model to emergency medical service system design, Decision Sciences 13 (1982) 416–439.

[13] M.S. Daskin, A maximal expected covering location model: Formulation properties and heuristic solution, Transportation Science 17 (1983) 48–69.

[14] M.S. Daskin, K. Hogan, C. ReVelle, Integration of multiple excess backup and expected covering models, Environment and Planning B: Planning and Design 15 (1988) 15–35.

[15] M. Desrochers, P. Marcotte, The congested facility location problem, Location Science 3 (1) (1995) 9–32.

[16] M. Gendreau, G. LaPorte, F. Semet, Solving an ambulance location model by tabu search, Location Science 5 (1997) 75–88.

[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, New York, 1989.

[18] K. Hogan, C. ReVelle, Concepts and applications of backup coverage, Management Science 32 (1986) 1435–1444.

[19] H.J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[20] C.R. Houck, J.A. Joines, M.G. Kay, Comparison of genetic algorithms, random restart and two-opt switching for solving large location–allocation problems, Computers & Operations Research 23 (6) (1996) 587–596.

[21] R.C. Larson, A hypercube queuing model for facility location and redistricting in urban emergency services, Computers & Operations Research 1 (1974) 67–95.

[22] V. Marianov, C. ReVelle, The queuing maximal availability location problem: A model for the siting of emergency vehicles, European Journal of Operations Research 93 (1996) 110–120.

[23] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1996.

[24] S.H. Owen, M.S. Daskin, Strategic facility location: A review, European Journal of Operational Research 111 (1998) 423–447.

[25] J. Repede, J. Bernardo, Developing and validating a decision support system for locating emergency vehicles in Louisville, Kentucky, European Journal of Operational Research 75 (1994) 567–581.

[26] C. ReVelle, D. Bigman, D. Schilling, J. Cohon, R. Church, Facility location: A review of context free and EMS models, Health Services Research 12 (1977) 129–146.

[27] C. ReVelle, Review, extension and prediction in emergency siting models, European Journal of Operational Research 40 (1989) 58–69.

[28] C. ReVelle, K. Hogan, The maximum availability location problem, Transportation Science 23 (1989) 192–200.

[29] J. Richardson, M. Palmer, G. Liepens, M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 191–197.

[30] C. Saydam, J. Repede, T. Burwell, Accurate estimation of expected coverage: a comparative study, Socio-Economic Planning Sciences 28 (2) (1994) 113–120.

[31] C. Saydam, M. McKnew, A Separable programming approach to expected coverage: An application to ambulance location, Decision Sciences 16 (1985) 381–398.

[32] D.A. Schilling, V. Jayaraman, R. Barkhi, A review of covering problems in facility location, Location Science 1 (1) (1993) 25–55.

[33] C. Toregas, R. Swain, C. ReVelle, L. Bergman, The location of emergency service facilities, Operations Research 19 (1971) 1363–1373.

[34] D Whitley, T. Starkweather, D.A. Fuquay, Scheduling problems and traveling salesman: The genetic edge recombination operator, in: Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, 1989, pp. 133–140.