Solving of the uncapacitated warehouse location problem using a simple genetic algorithm

Jozef Kratica[1] , Vladimir Filipović[2], Vesna Šešum[1] and Dušan Tošić[2]

Abstract

The uncapacitated warehouse location problem is considered. Since it belongs to the class of NP complete problems, we use the genetic algorithms in the solving of this problem. Genetic algorithms are rooted in the mechanisms of the evolution and natural selection. They are relatively general and practicale way for the finding a suboptimal solution (heuristic) in the problems of optimization.

According to the uncapacitated warehouse location problem, we should find provision plan with minimal total cost. The storage cost for every warehouse and the cost of shipment from every warehouse to an arbitrary customer are known.

We use simple genetic algorithm for the solving of uncapacitated warehouse location problem.. The item-code is represented by the binary array of indicators denoting the inclusion of warehouse into provision plan.

This approach seems to be a good compromise between the quality of solution and execution time. The improvements are possible by introducing of other selection and crossover operators.

Keywords: Uncapacitated warehouse problem, NP complete problems, Genetic algorithms

1. Introduction

1.1. The uncapacitated warehouse problem

Suppose that there are n warehouses and m clients. Let c_i ($1 \le i \le n$) be a storage cost of merchandise in a warehouse i, and h_{ij} ($1 \le i \le n$, $1 \le j \le m$) a cost of shipment from a warehouse i to client j. We should find the warehouses included into the provision plan and the way of shipment from these warehouses to clients. The problem is to find a provision plan that has a minimal total cost (the sum of storage cost and cost of shipment). (See: [2],[3] and [4]).

Denote with x_i ($x_i \in \{0,1\}$, i = 1, …, n) i-th element of binary array indicating that the warehouse i is included, or not, into the provision plan. Let y_{ij} be the quantity of shipment from the warehouse i to the customer j. It is necessary to find:

$$\min\left(\sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij} y_{ij}\right)$$

according to the following conditions:

$$\sum_{i=1}^{n} y_{ij} = 1, \quad \text{for } j = 1, 2, \ldots, m;$$

$$y_{ij} \le c_i x_i, \quad \text{for } i = 1, 2, \ldots, n; \text{ and } j = 1, 2, \ldots, m.$$

This problem belongs to the class of NP complete problems and its optimal solution could be obtained only by the algorithms with exponential execution time. These algorithms are convenient only for the small values of m and n ($m,n \le 20$), but they are inefficient, and practically useless, for the greater values of m and n.

[1] Katedra za Matematiku, Mašinski Fakultet u Beogradu, 27. marta 80, 11000 Beograd
[2] Matematički fakultet u Beogradu, Studentski trg 16, 11000 Beograd

So, to solve the uncapacitated warehouse problem for larger values of m and n (m,n > 20), it is necessary to use suboptimal, heuristic algorithms. Some of these algorithms are described in [3], [4] and [8].

1.2. Genetic algorithms

Genetic algorithms (developed in the sixties) are direct, random search algorithms based on the model of biological evolution ([1], [6]). Consequently, the terminology used in genetic algorithm is borrowed from the natural genetic. These algorithms rely on the collective learning process within a population of individuals. Each of the individual represents a search point in the space of potential solutions of a given optimization problem. The population evolves towards increasingly better regions of the search space by means of randomized processes of selection, mutation and recombination (crossover). The selection mechanism favors individuals of better objective function value to reproduce more often than worse ones when a new population is formed. The recombination allows the mixing of parental information when this pass to their descendants, and mutation introduces innovation into the population. Usually, the initial population is randomly initialized and evolution process is stopped after predefined number of iterations. For more detailed explanations see [1], [6], [9] and [10].

2. The way of solution by genetic algorithms

2.1. Coding

To solve the given problem, it is natural to use the binary coding. Every bit in the item-genetic code with the value 1, denotes a specific warehouse included into the provision plan, and 0 denotes that it is not included.

Example 1: For n=5, genetic code 01011 denotes that warehouses 2, 4 and 5 are included into provision plan, and 1 and 3 are not.

In our implementation of the genetic algorithm an item-genetic code is divided in 32-bit words (unsigned long integers), but we do not use all bits in the last word. As we can see in the example 1, for n=5 every item has a genetic code represented by a 32-bit word, but the bits 0-4 are only used (this is the simple case where only one word is used, because n < 32), while bits 5-32 have not any function.

By using genetic code we obtain an array $\{x_i\}$ (i=1, 2, .. , n) indicating the warehouses included in the provision plan. After that, the following procedure is used.

Since the quantity of merchandise delivered from arbitrary warehouse is not limited, if all of clients choose minimal cost of shipments, the whole cost is minimal. In the choosing procedure for a specific client, only warehouses included in the provision plan ($x_i = 1$) are considered.

That fact implies minimization only in the warehouse choosing procedure. If the warehouse choosing procedure is fixed, the minimal total cost is obtained easily, because the shipment is done from the most favorable warehouse for the client.

Example 2: Suppose that we have five warehouses with the storage costs, successively, 5, 2, 7, 3, 4; and three clients. Let the transport cost matrix is:

1	1.5	0.8	2	1.6
2	1.7	1	1.5	1.3
1.5	1.3	2	1.5	1

If warehouses 2, 4 and 5 are included in the provision plan, as in example 1, the storage cost is 2 + 3 + 4 = 9. The minimal transport costs for clients are, successively: 1.5 (minimum from 1.5, 2 and 1.6), 1.3 (minimum from 1.7, 1.5 and 1.3) and 1 (minimum from 1.3, 1.5 and 1). The total cost of that supplying plan is 9+1.5+1.3+1 = 12.8. The obtained cost is minimal for this choose of warehouses, but it is not an optimal supplying plan.

The optimal supplying plan in example 2 is obtained only if the warehouse 2, with the storage costs 2, is included. The transport costs are, successively, 1.5, 1.7 and 1.3. The total cost of that plan is 2+1.5+1.7+1.3 = 6.5.

2.2. Genetic algorithm parameters

We test our program for the solution of uncapacitated warehouse problem on the population of 20 items, and the maximal number of generations 1000.

We used simple genetic algorithm: simple selection, one-point crossover and simple mutation. As it is a minimization problem, the fitness of every item is reversed to its objective value. Formula for that scaling into the interval (0,1) is

(**max**-**p**)/(**max**-**min**)

where **max** and **min** are maximal and minimal item objective values in current generation, and **p** is a particular item objective value.

In the simple selection, chances for passing of given item into next generation, are directly proportional to its fitness. The best items are frequently passed into the next generation.

The choosing a pair of items in the one-point crossover is random. After that, for given items, we choose the crossover position and exchange bits behind that position. In our implementation crossover rate is 0.85. This represents a ratio of the crossover item´s number, and the population size. The mutation rate is 0.005 for the large instances (A - C), and 0.01 for the small instances (71 - 134). For the small instances number of bits in item genetic code is smaller, and mutation rate of 0.005 is not enough to prevent premature convergence. For more detailed explanations about simple genetic algorithm see [1], [6], [9] and [10].

The only change in our implementation of the simple genetic algorithm is: best item in the current generation passes directly to the next generation without select, crossover and mutation.

3. Results

The described genetic algorithm is written in the programming language C. The testing is done by using the PC compatible computer I80486 at 133Mhz.

Test examples are standard for testing the solution obtained by genetic algorithm. They can be found on the Internet FTP address: *mscmga.ms.ic.ac.uk* (WWW address is: *http://mscmga.ms.ic.ac.uk/*). For more detailed explanations about these test examples, for uncapacitated and capacitated warehouse problem, see [2] and [3].

Since genetic operators of selection, crossover and mutation are undeterministic, every test example has to run multiple times. We have tested it 20 times. Results are given in the table 1.

Test	N	M	Opt.	<0.2 %	<1%	<5%	>5%
71	16	50	19	0	0	1	0
72	16	50	20	0	0	0	0
73	16	50	18	2	0	0	0
74	16	50	20	0	0	0	0
101	25	50	16	0	4	0	0
102	25	50	20	0	0	0	0
103	25	50	9	0	10	1	0
104	25	50	20	0	0	0	0
131	50	50	9	0	7	4	0
132	50	50	17	1	1	1	0
133	50	50	7	2	7	4	0
134	50	50	15	0	2	0	3
A	100	1000	1	1	0	8	10
B	100	1000	3	2	0	15	0
C	100	1000	1	1	2	16	0

Table 1.

The first column in the table 1. represents the ordinal number of that test in [2] and [3]. The next two columns represent dimensions of the problem (N and M). The columns 4, 5, 6, 7 and 8 are, successively:

- the number of the obtained optimal solution,
- the number of solution with the error less than 0.2% from optimal solution,
- the number of solution with error from optimal solution was between 0.2% and 1%,
- the number of solution with the error between 1% and 5% in comparison to the optimal solution,
- the number of solution with the error greater than 5% from optimal solution.

Note that at examples for N=16 (M=50) and N=25 (M=50), in almost every case, program obtains optimal solution. At examples for N=50 (M=50), in about half

and more cases, we obtain optimal solution. At examples with large values for N and M (N=100, M=1000) only in a few cases the optimal solution is obtained.

The execution time in examples with the small values of M and N (test examples 71 - 134) was about 10 seconds, while for large values of M and N (test examples A - C) it was about 3 minutes.

4. Conclusion

This is the first implementation of genetic algorithms for solving the uncapacitated warehouse location problem. The binary coding of warehouse, included in the provision plan, is especially good combination with nature of genetic algorithms. That fact later contributes to successful use of genetic operators: selection, crossover and mutation.

The main advantage of this implementation is relatively short execution time. It is also valid for the examples with large values of M and N, where the other algorithms ([3], [4], [8]) usually do not give good solution or have very large execution time.

In our implementation the search space size, for examples *71-74* is 2^{16}=65536, for examples *101-104* is 2^{25}=3.3*10^7, for examples *131-134* is 2^{50}=10^{15}, and in examples with large values (A-C) it is even 2^{100}=10^{30}. Using this implementation for 1000 generations by 20 items (20 000 calls to objective function) we get the optimal solution at least once in all test examples. Note that in large examples we search about 10^{-24} of search space size.

The improvement of this implementation is possible by applying some others genetic operators for selection, crossover and mutation, described in [1], [6], [9] and [10].

Further improvements could be obtained by combining this method with some other methods for solving given problem (see: [2], [3], [4] and [8]), Also we can use some hybrid techniques mentioned in [7].

The approach applied to the uncapacitated warehouse problem could be applied in solving of capacitated warehouse location problem, too. Of course some modifications of our impementation are necessary.

The numerous C-functions from this implementation could be used for solving the other NP-complete problems.

5. References

[1] *Beasley, D., Bull, D.R., Martin, R.R.,* **An Overview of Genetic Algorithms,** University Computing 15 (4), pp 170-181, 1993.

[2] *Beasley, J.E.,* **An algorithm for solving large capacitated warehouse location problems**, European Journal of Operational Research, volume 33 (3), pp 314-325, 1988.

[3] *Beasley, J.E.,* **Lagrangean heuristic for location problems**, European Journal of Operational Research, volume 65 (3), pp 383-399, 1993.

[4] *Brandeau, M.L., Chiu, S.S.,* **An overview of representative problems in location search**, Management Science, volume 35 (6), pp 645-674, 1989.

[5] *De Jong, K.E., Spears, W.M.,* **Using Genetic Algorithms to Solve NP-Complete Problems**, Proceedings of the Third International Conference on Genetic Algorithms, pp 124-132, 1989.

[6] *Goldberg, D.E.,* **Genetic Algorithms in Search, Optimization and Machine Leraning**, Addison-Wesley, Reading, 1989.

[7] *Kido, T., Kitano, H., Nakanishi, M.,* **A Hybrid Search for Genetic Algorithms: Combining Genetic Algorithms, TABU Search, and Simulated Annealing**, Proceedings of the Fifth International Conference on Genetic Algorithms, pp 641, 1993.

[8] *Panneerselvam, R., Balasubramanian, K.N., Thiagarajan, M.T.* **Models for warehouse location problem**, International Journal of Management and Systems, volume 6 (1), pp 1-8, 1990.

[9] *Ribeiro-Filho, J.L., Treleaven, P.C., Alippi, C.,* **Genetic-Algorithm Programming Environments,** IEEE Computer, June 1994., pp 28-43

[10] *Srinivas, M., Patnaik, L.M.*, **Genetic Algorithms: A Survey**, IEEE Computer, June 1994., pp 17-26

[11] *Vignaux, G.A., Michalewicz, Z.*, **A Genetic Algorithm for the Linear Transportation Problem**, IEEE Transactions on Systems, Man, and Cybernetics, volume 21 (2), pp 445-452, 1991.