Solving the Assignment problem using Genetic Algorithm and Simulated Annealing

Anshuman Sahu, Rudrajit Tapadar.

***Abstract*—The paper attempts to solve the generalized "Assignment problem" through genetic algorithm and simulated annealing. The generalized assignment problem is basically the "N men- N jobs" problem where a single job can be assigned to only one person in such a way that the overall cost of assignment is minimized. While solving this problem through genetic algorithm (GA), a unique encoding scheme is used together with Partially Matched Crossover (PMX). The population size can also be varied in each iteration. In simulated annealing (SA) method, an exponential cooling schedule based on Newtonian cooling process is employed and experimentation is done on choosing the number of iterations (m) at each step. The source codes for the above have been developed in C language and compiled in GCC. Several test cases have been taken and the results obtained from both the methods have been tabulated and compared against the results obtained by coding in AMPL.**

***Index Terms*—Assignment problem, Genetic Algorithm, Newtonian cooling schedule, Partially Matched Crossover (PMX), Simulated Annealing.**

I. Introduction

The Assignment model, as discussed in different text-books of Operations Research, can be paraphrased as: "Given N men and N machines, we have to assign each single machine to a single man in such a manner that the overall cost of assignment is minimized." To put it mathematically, let us define the following symbols:

i →row number denoting i^{th} man i ε [1, N]

j →column number denoting j^{th} machine j ε [1, N]

$C[i][j] \rightarrow$ cost of assigning j^{th} machine to i^{th} man

$X[i][j]$ = 1 if j^{th} machine is assigned to i^{th} man

= 0 otherwise

The problem can be formulated as:

Minimize the total cost function $\sum_{i=1}^{N}\sum_{j=1}^{N} C[i][j]X[i][j]$

Subject to the following constraints:

$$\sum_{i=1}^{N} X[i][j] = 1 \quad \forall \text{ j=1, 2… N} \tag{1}$$

$$\sum_{j=1}^{N} X[i][j] = 1 \quad \forall \text{ i=1, 2… N} \tag{2}$$

$$X[i][j] = 1 \text{ or } 0 \tag{3}$$

The Hungarian mathematician D.König proved an essential theorem for the development of the "Hungarian method" to solve this model. The problem can also be formulated as an integer-programming model and solved by techniques such as "Branch-and-Bound technique". Reference [1] states that the Hungarian algorithm for solving the assignment model is more efficient than branch-and-bound algorithms. This paper attempts to solve the same model using two non-traditional techniques: Genetic Algorithm and Simulated Annealing. It is basically an experimental investigation into the various parameters affecting these two algorithms and adapting them to our own problem. These two approaches are discussed one by one.

II. GENETIC ALGORITHM APPROACH

Genetic algorithms (GA) are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. They were first envisioned by John Holland and were subsequently developed by various researchers. Each potential solution is encoded in the form of a string and a population of strings is created which is further processed by three operators: Reproduction, Crossover, and Mutation. Reproduction is a process in which individual strings are copied according to their fitness function (Here the fitness function is taken to be the total cost function). Crossover is the process of swapping the content of two strings at some point(s) with a probability. Finally, Mutation is the process of flipping the value at a particular location in a string with a very low probability. A more comprehensive treatment of GA can be found in [2], [3], [4].

Now, for adapting GA to our problem, it is necessary that we develop an encoding scheme. Consider the case when N=3 and let us presume that machine M1 is assigned to man m1, machine M2 to man m2, and machine M3 to man m3 as shown:

.

Manuscript received March 7, 2006.

Anshuman Sahu is a senior undergraduate student with a major in Production and Industrial Engineering in Motilal Nehru National Institute of Technology, Allahabad, India. (e-mail: anshumnnit@gmail.com).

Rudrajit Tapadar is a junior undergraduate student with a major in Computer Science and Engineering in Motilal Nehru National Institute of Technology, Allahabad, India. (e-mail: rudrajit.tapadar@gmail.com).

		Machine→ M1	M2	M3
	m1	1	0	0
Man	m2	0	1	0
↓	m3	0	0	1

Figure 1. A sample assignment.

Consider the first column: 100 which is equivalent to 4 in base 10 representation. Similarly the other two columns decode to 2 and 1 respectively. Hence, the above assignment can be encoded as <4 2 1>. A quicker insight leads us to the observation that the each permutation of <4 2 1> i.e., <1 2 4>, <1 4 2>, <2 1 4>, <2 4 1>, <4 2 1>, and <4 1 2> is a possible solution. As the total number of solutions possible to this particular problem are 3! =6, we can easily conjecture that in case of N men, N machine, the total number of solutions possible is N! and our task is to select the best string (the one with minimum total cost). As our encoding scheme also generates N! strings, therefore it is correct and there is one to one correspondence between each possible solution and each string. It is also evident that each component (value at each position) in the string can be uniquely expressed as 2^r where r is a positive whole number varying from 0 to N-1. As the powers of 2 increase rapidly, a more compact way of encoding would be to express the component 2^r simply as r. This is easier to write and saves space when N is high.

After encoding of the string, the population selection for crossover is done by "Binary tournament selection" method. Here s=2 strings are randomly chosen and compared, the best one being selected for parenthood. This is repeated M times where M is the size of the population. Reference [4] also cites a method for generating the parent strings which are then ready for crossover. Here simple crossover will not work; instead we choose the method of Partially Matched Crossover (PMX) which was initially developed for tackling the "Traveling Salesman Problem" [2]. The concept of PMX can be understood by considering an example:

Suppose we want to have crossover between two permutations of the string <1 2 3 4 5> i.e., <1 3 4 2 5> and <2 1 3 5 4>. Two random numbers are generated between 1 and L where L is the length of the string (L=5 in this case). Suppose the crossover points have been choosen as shown below:

1 3 | 4 2 | 5

2 1 | 3 5 | 4

Where the dashed positions show the chosen points. Now PMX defines the following scheme for interchangeability:

3 ↔1 4 ↔3 2↔5 implies 1↔4 and 2↔5

Now the portion between the selected crossover points is swapped and the rest of the values are changed according to the above rule (this means 1 in the portion outside the two crossover points is replaced by 4 and 2 in the portion outside the two crossover points is replaced by 5). So the two children strings generated are:

4 1 3 5 2

5 3 4 2 1

Which are again valid permutations of <1 2 3 4 5>. After Crossover, we have a family of parent population and children population out of which we are to select the population for next iteration. Here we have a choice of altering the population size at each iteration. We must maintain the diversity in population or else it may lead to premature convergence to a solution which may not be optimal. One method of selecting the population may be to arrange the entire population in ascending order of their objective function value (the string that decodes to lowest total cost of assignment will have the highest objective function value) and choose a predetermined number of individual strings from each category i.e., from those that are above average, from those around the average, and from those below the average. This threshold can be set by using the concept of mean and standard deviation applied to the population. For instance, if we assume the string values to be normally distributed with mean value μ and standard deviation σ, we divide the population into four categories: those having values above $\mu + 3*\sigma$, those having values between $\mu + 3*\sigma$ and μ, those having values between μ and $\mu - 3*\sigma$, and those having values less than $\mu - 3*\sigma$. In this way the diversity in population is maintained. Another aspect is that the string with the best objective function value at each iteration is stored in a separate array and subsequently compared with the best string of the population at next iteration. In this way, the best string cannot escape. Also note that we are not using mutation but a slight variant of it (Inversion) by choosing two random spots in a string and swapping the corresponding values at that position. Inversion is allowed only when the sum of the costs at these positions before swapping is greater than the sum of costs associated with these positions after swapping.

Thus, if we want to swap in the string <1 2 3 4> at say second and third positions, it will only be allowed if cost of <1 2 3 4> is greater than cost associated with <1 3 2 4>.

The program was developed for the test problem given in [1]. Two cases were implemented: one in which Inversion was used and another in which Inversion was not used. In both the cases, the answer converged to the final optimum value. On an average, there was not much difference in the number of iterations required to reach the final value in both the cases. The observations are plotted in the table as shown below:

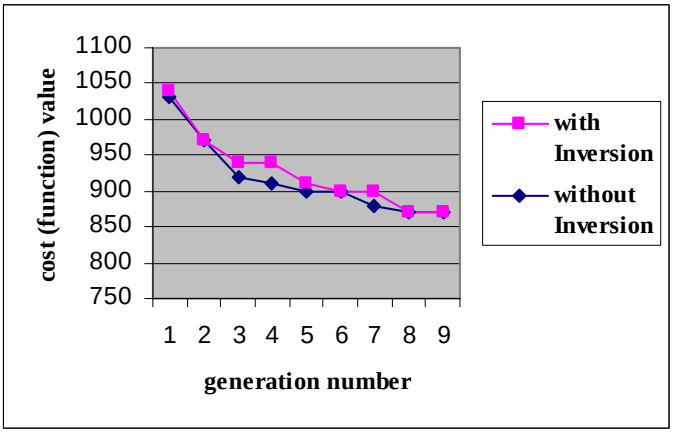

Figure 2. Graph showing convergence to the global minimum in case of both inversion and without inversion.

On an average, the time taken was 0.01s measured on a standard desktop with processor Intel Pentium 4, 2.40 GHz. The population size at each generation was kept equal to 20.

III. SIMULATED ANNEALING APPROACH

Simulated Annealing is another non-traditional method which was originally developed by S. Kirkpatrik, C.D. Gelatt, Jr., and M.P. Vecchi [5]. The simulated annealing procedure simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. It is a point-by-point method. The algorithm begins with an initial point and a high temperature T. A second point is taken at random in the vicinity of the initial point and the difference in the function values (ΔE) at these two points is calculated. The second point is chosen according to the Metropolis algorithm which states that if the second point has a smaller function value, the point is accepted; otherwise the point is accepted with a probability exp (-ΔE / T). This completes one iteration of the simulated annealing procedure. In the next generation, another point is created at random in the neighborhood of the current point and the Metropolis algorithm is used to accept or reject the point. In order to simulate the thermal equilibrium at every temperature, a number of points (m) is usually tested at a particular temperature before reducing the temperature. The algorithm is terminated when a sufficiently small temperature is obtained or a small enough change in the function values is found. A detailed description of this can be found in [4].

While using this algorithm in our case, we represented each possible solution by the string as developed previously in the case of genetic algorithm. E refers to the function value (the total cost of assignment for a particular string). We have employed the scheme of Newtonian cooling wherein the temperature at each generation is determined according to the law: $T_i = T_0 * \exp(-\tau)$ where T_i is the temperature at i^{th} generation, T_0 is the initial temperature and τ is a suitable constant (τ is initially taken 0 when temperature equals T_0 and is incremented by a factor "increment" at each stage). Now consider the task of randomly generated valid strings: two techniques are being employed. Suppose we have the string <1 2 3 4> and we want to produce another random permutation of these 4 numbers. The first method is to slide each number by a random number generated between 1 and L (L not included) where L is the length of the string. Thus, assuming that the random number generated is 2, the string <1 2 3 4> gets transformed to <3 4 1 2>. The second method of generating a valid permutation is two choose two positions at random in the string and swap the values at those points. We search for the potential solution in two regions: first we search in the region of strings created by the above first method. When the answer converges to a particular value, we store the corresponding string in a separate array. Then we proceed with our search again in the region of strings created by the second method. Once again, we converge to another string and this string is compared with the string which was initially stored in a separate array. The minimum of these two (the one with lesser function value) is selected as the final answer.

The important parameters affecting simulated annealing are the number of iterations (m) at each step and the cooling schedule. The total number of iterations is proportional to m as well as the rate of change of temperature. The cooling schedule is based on Newton's law of cooling. This model of cooling can be compared to the discharge of an initially charged capacitor in a RC circuit as they both follow exponential decay law. For all practical purposes, it is assumed that the capacitor is fully discharged at t=5*RC. Hence, in our schedule we also ran our program from Tmax=700 to Tmin around 700*exp (-5), keeping the number of iterations m fixed (=50). Tmax is generally computed by calculating the average of function values at several points. The program was run on a standard desktop with processor Intel Pentium 4, 2.40 GHz and the test case considered was the one given in [1]. The results obtained have been plotted as shown in figure 3 as shown below:

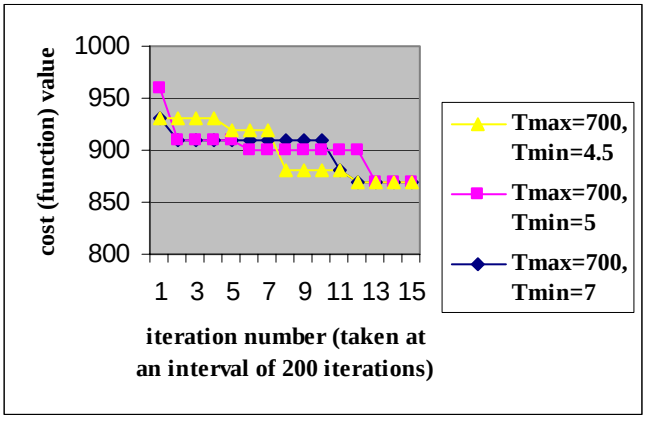

Figure 3. Program run for various schedules (m constant)

The average time taken was 0.051s. Now m at each step was changed, decreasing it from an initial value of 100 till a minimum value (=20 in our case) was reached. It was observed that the program converged to the minimum value at lesser total number of iterations. This is shown below:

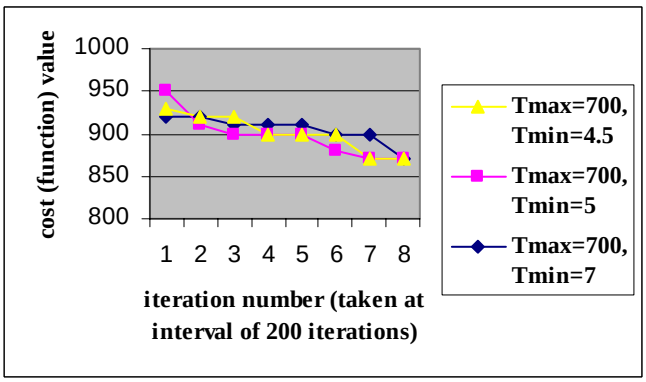

Figure 4. Program run for various schedules (m varying)

Reference [1] reports to have solved the above test problem

in 0.09s of IBM 370/168 time. The problem was also coded in AMPL with MINOS 5.5 as the solver and it took 0.03125s on the standard desktop mentioned earlier. While solving the problem using Genetic Algorithm, the average time taken was 0.01s while the time taken for solving it using Simulated Annealing was 0.05s (The time was noted on a standard desktop with processor Intel Pentium 4, 2.40 GHz).

IV. Conclusion

An experimental investigation into solving the Assignment model using Genetic Algorithm and Simulated Annealing is presented. Various parameters affecting the algorithms are studied and their influence on convergence to the final optimum solution is shown.

Acknowledgment

The Authors would like to thank Dr. Sanjeev Sinha, Asst. professor, Dept. of Mechanical Engineering, MNNIT, India for his invaluable advice and guidance. The Authors would also like to thank their friends with whom they discussed their ideas which sometimes led to many new insights.

References

[1] Billy E. Gillett, *Introduction to Operations Research A Computer-Oriented Algorithmic Approach.* Tata McGraw-Hill Publishing Company Limited, New Delhi (Copyright © 1976 by McGraw-Hill, Inc., New York), TMH EDITION 1979, ch. 3, ch.4.

[2] Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, Mass.: Addison-Wesley, 1989.

[3] Kalyanmoy Deb, *Optimization for Engineering Design Algorithms and Examples.* Prentice-Hall of India Private Limited, New Delhi, 1995, ch. 6.

[4] Fred Glover, Gary A. Kochenberger, Ed. *HANDBOOK OF METAHEURISTICS.* ©2003 Kluwer Academic Publishers New York, Boston, Dordrecht, London, Moscow (Print ISBN: 1-4020-7263-5, ebook ISBN: 0-306-48056-5).

[5] Kirkpatrick, S., Gelatt, Jr., C.D. and Vecchi, M.P. (1983) "Optimization by simulated annealing", *Science,* **220**, 671-680.