A Modified Heuristic Procedure for NP-Complete Problems Supported by Genetic Algorithms

Najla Akram AL-Saati
dr.najla_alsaati@uomosul.edu.iq
College of Computer Sciences and Mathematics
University of Mosul, Iraq

Received on: 16/07/2003 **Accepted on: 07/10/2003**

Abstract

This work is based on the process of modifying an intelligent heuristic rule used in solving NP-Complete problems, where a study and a modification of a Flow Shop assignment heuristic has been carried out to solve a well-known classic Artificial Intelligent problem, which is the traveling Salesman problem. For this modification to be carried out successfully, the problem's mathematical formulation had to be studied carefully and the possibility of reformulating the problem to be more suitable for the heuristic procedure. This may require some changes in the heuristic procedure itself, these adjustments were due to the noticeable differences like the symmetric property present in the traveling salesman problem environment and some other differences.

Genetic Algorithm is added to improve the results obtained by the used heuristic, where the use of crossover and mutation procedures will provide better chances for the near optimal solution to be improved towards optimal solutions.

The test problem is made on cities that lie on the regular square grid, which simplify the calculations of distance traveled between any two cities. Programs were written using C programming language, and timers were used to measure the elapsed time of calculations in order to assess the efficiency of the program.

Keywords: Heuristic, NP-Complete Problems, Genetic Algorithms.

اجراء حدسي محسن لمسائل الـ (NP-Complete) مدعوما بالخوارزميات الجينية

نجلاء اكرم الساعاتي

كلية علوم الحاسوب والرياضيات / جامعة الموصل

تاريخ استلام البحث: 2003/07/16 تاريخ قبول البحث: 2003/10/07

الملخص

يرتكز هذا العمل على عملية تحوير قانون حدسي ذكي يستخدم في حل المسائل الـ (NP-Complete) وقد تمت دراسة احد القوانين الحدسية لمسائل التعيين من نوع(Flow Shop Problem) وتحويره ليستخدم في حل إحدى مسائل الذكاء الاصطناعي الشائعة والتقليدية وهي مسالة البائع المتجول (Traveling Salesman Problem). ومن اجل إنجاح هذا التحوير يجب ان تتم دراسة التشكيل الرياضي للمسألة بعناية ودراسة إمكانية إعادة تشكيل الصيغة الرياضية بما يتلاءم مع طبيعة القانون الحدسي. وقد يتطلب ذلك تقديم تحويرات بسيطة في القانون الحدسي نفسه بسبب بعض الاختلافات الملحوظة كالتناظر (Symmetry) الموجود في بيئة مسالة البائع المتجول وغيرها من الاختلافات.

ولتحسين الحلول الناتجة من استخدام الحدس تضاف البرمجة الوراثية في هذا العمل، حيث ان استخدام التبادل والطفرات الوراثية سيتيح للحل الحدسي الجيد خطا اوفر للتحسن والاقتراب من الحلول المثلى.

اجريت عملية الفحص على مدن واقعة على الشبكة المربعة الاعتيادية والتي تبسط العمليات الحسابية للمسافات المجتازة بين اي مدينتين. وقد تمت كتابة البرامج باستخدام لغة (C) البرمجية فضلا عن استخدام الموقت البرمجي لقياس الوقت المنصرم لتنفيذ العمليات الحسابية وذلك لقياس كفاءة البرنامج.

كلمات مفتاحية: الحدس، مسائل (NP-Complete)، الخوارزميات الجينية.

1- Introduction:

The NP-Complete class of problem types covers a wide range of assignment, scheduling, transportation and other types of problems. A problem is said to be NP-Complete if and only if it is NP-Hard and it is in NP [6]. This class of problems is very hard to solve because it requires a non-polynomial number of solution steps and their search space increase exponentially as the problem becomes larger, leading at some point to what is known as combinatorial explosion.

Due to the importance of this class of problems many heuristics have been proposed to give good satisfying solutions in a reasonable time. Heuristic search methods are most often based on maximizing or minimizing some aspect of the problem. In many instances, their use may reduce the problem space sufficiently so that an analytical or simulation procedure may be applied, sometimes, the heuristics themselves may be sufficient to produce an acceptable feasible solution.

A very well known problem of this class is the traveling salesman problem (TSP), the American mathematician Hassler Whitney posed the TSP in 1934. Whether it has yet been solved depends on exactly what one means by solve. The problem may be stated as follows:

A commercial traveler has n customers, all in different towns; he wishes to visit them all in turn, starting and ending his journey at the same starting town and not passing through an intermediate town twice. Knowing the distance between every pair of towns, in what order should he make the visits so as to travel the least possible total distance?

For some reasons this problem has become a classic NP-Complete problem[6]. Almost every applied mathematician interested in operations research has tried his hand at solving it. A large number of algorithms has been developed and there is a remarkable number of very ingenious translations of this problem into other problems that may be easier to solve.

The TSP has been studied within the framework of linear programming, but the method is somewhat cumbersome in application and requires a great deal of execution time. Enumeration of all routes *n* is a discouraging process for a problem of any size, since there are (*n-1*!) possible routes. Doubling the size of the problem from 5 to 10 cities multiplies the number of routes by about 15,000.

All this led to the development of continuously improving optimal solution methods. Certain algorithms employing branch and bound techniques have been tried and appeared to be efficient for some problems; beginning generally with complete paths, keeping track of the shortest path found so far, give up exploring any path as soon as its partial length becomes greater than the shortest path found so far. Using this technique, the shortest path is still guaranteed to be found. Although this algorithm is more efficient than a complete search through the tree, the computational time involved is still unpredictable and increases very rapidly with N, this is due to the fact that it requires exponential time, and for this reason it is inadequate for solving large problems.

Many algorithms have been proposed recently to solve this problem in an attempt to find more optimal solutions in reasonable time limits (Padberg and Rinaldi [10] and Gr̥tschel and Holland [5]), procedures capable of handling small problems as well as some efficient heuristics. (Gendreau, Hertz and Laporte [4], Zweig [12] and Renaud, Boctor and Laporte [11]) to handle larger problems.

2- Heuristic Solution Methods:

Heuristic problem solving is not a solution in the sense that the simplics method of linear programming is, but rather it is a philosophy of seeking out a method or methods that might produce a solution to a particular problem. Heuristics are used as techniques for improving the efficiency of a search process,

possibly by sacrificing claims of completeness, meaning that the optimality of the solution is not always guaranteed.

The word heuristic is often used not just to describe cases where a solution might not be found, but also to describe cases where we want to find the best solution (according to some way of measuring bestness). A heuristic might help to find solutions which are good, but perhaps not the very best that can be.[2]

In general, heuristics are information about the likelihood that a specific node is a better choice to try next rather than another; they are simply rules that qualify the possibility that a search is proceeding in the correct direction. They simplify the environment of the decision-maker by permitting him to make decisions quickly without considering the number of ways that each decision can be made. They limit the search by reducing the number of alternatives to a manageable size.

Whereas analytical procedures are based on deductive reasoning supported by mathematical proofs and known properties, heuristic methods are based on inductive inferences related to human characteristics of problem solving, such as creativity, insight, intuition, and learning. A particular heuristic is followed because it promises, intuitively or from experience, to help in the search for an acceptable solution, and in the process a better rule is discovered, then the old one is discarded. So while heuristic problem solving involves the use of currently accepted rules, it also involves a search for even better rules to replace them.

Using the heuristic search procedure based on rules from accumulated experience is more efficient than the analytical procedure in a majority of cases. This is so because the heuristic method uses all information available at each stage of the search in order to formulate a future strategy. The information to be sought on each successive trail is determined by the information previously obtained; this enables a convergent search for the solution. On the other hand, the analytical method has rigid data

requirements, as is the case with most standard operations research techniques.

Desirable characteristics of heuristics:

- Execution in a reasonable computational time.
- Solutions that are close to optimality on the average.
- Small probability of any solution being far below optimality.
- Simplicity of both design and computational results.

Heuristic methods rely on intuitive or empirical rules that have the potential to determine an improved solution relative to the current one. Actually, heuristics are search procedures that intelligently move from one solution point to another with the objective of improving the value of the model objective. When no further improvements can be achieved, the best-attained solution is the approximate solution to the model.

3- The Modified Heuristic Solution:-

Scheduling in a flow shop requires determining the sequence in which available jobs will be processed. The flow shop problem (FSP) is to assign integer values to variables x_{ij}, where x_{ij} is the starting time of job i on machine j, $j = 1, ..., n$. Given the time t_{ij} that it takes to complete the work of job i on machine j, the problem is to schedule the jobs on each machine so that the total time for the completion of all the jobs is minimum. Because $(x_{in} + t_{in})$ is the time at which job i is completed on machine n, the maximum of these numbers is the time at which the latest job is completed. It is that time which is to be minimized. The criterion is:

$$\textbf{\textit{Minimize } } f(x) = \max (t_{in} + x_{in})$$

The straightforward approach of the enumeration methods works well for very small problems but it rapidly grows beyond the bounds of practicality for even today's high-speed computers as the number of jobs increases. Complete enumeration of a problem involving only twenty-five jobs requires calculating the

objective function value for 10^{25} different sequences. Since there have been less than (3.2×10^{14}) year since the creation of the universe, it is clear that even a very powerful computer would not be of much help if someone wishes to solve a (25 x 25) assignment problem by exhaustive enumeration methods. As a result, heuristic methods have been developed in order to reduce the number of sequences to be considered, and determine a good (near optimum), but not necessarily optimal solutions.

Nawaz, Enscore, and Ham [8] proposed a new heuristic approach called (NEH). Working with the sum of the processing times for each job, they first select the two jobs with the greatest total processing times. The two jobs are ordered in a partial sequence that gives the best Processing time for the partial sequence. The relative positions of the two jobs with respect to each other are fixed for the remaining steps of the procedure. The unscheduled job with the highest total processing time is tried in every possible position in the existing partial sequence, creating a new partial sequence with minimum Processing time (Figure 1). This process is repeated until all jobs are assigned to sequence positions [8].

While in the case of the traveling salesman problem, the problem is subjected as a matrix of c_{ij}'s in which each c_{ij} represents the distance from city *i* to city *j*. The salesman must start from the first city, visit each of the other cities once, return to the first city, and he is to minimize the distance of his tour. So far the problem looks exactly like an assignment problem; there are, however, some subtle differences.

First, in the assignment problem, job *i* could be assigned to worker *i*, while it makes no sense for the salesman to go from city *i* to city *i*. The second distinction is that in an assignment problem there is nothing to prevent us from assigning the first worker to the second job and the second worker to the first job; but if the salesman goes from the first city to the second city, and then return from the second city to the first, he never gets to the third city. Furthermore, if he goes from the first city to the

second and from that to the third, he may not return to the first city because then he never gets to the fourth city. Thus, cyclic solutions are not permitted. This prohibition on cyclic solution is easy to verbalize but hard to formalize mathematically.

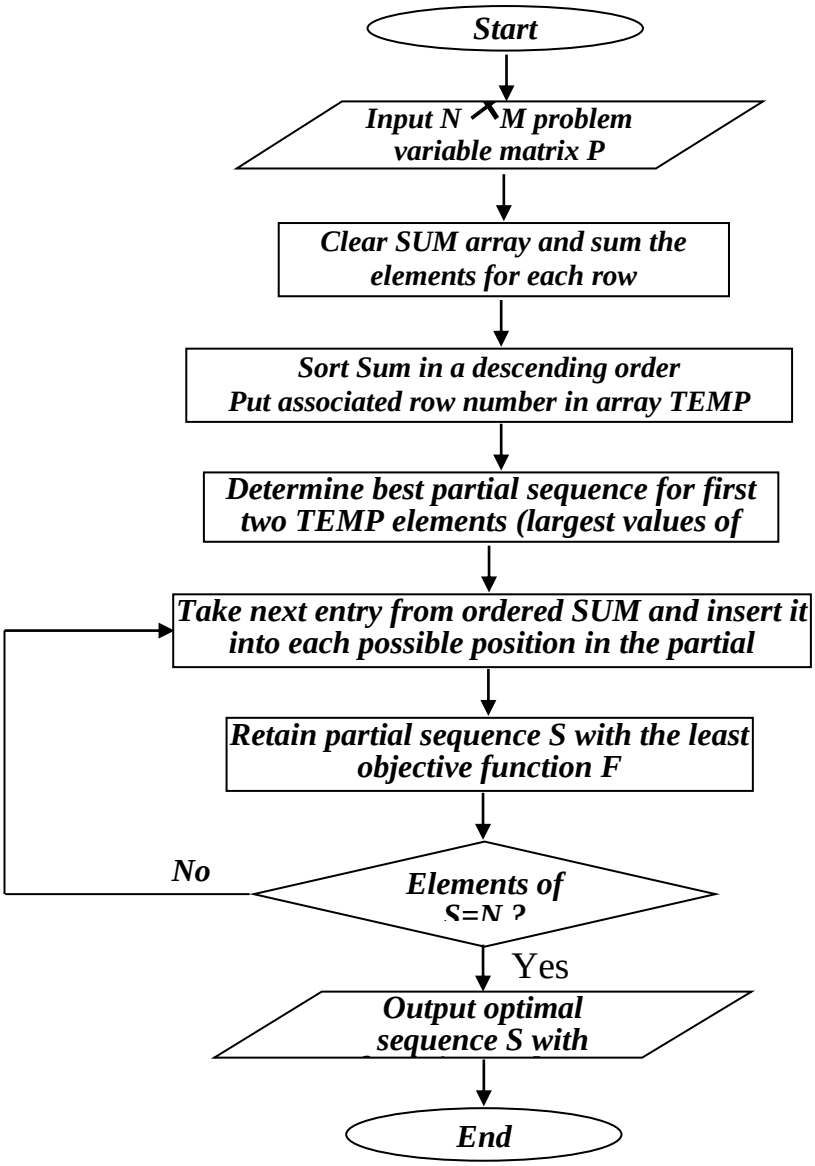

Figure-1 The NEH Heuristic Procedure

The Traveling Salesman Problem is Probably the best-known problem in combinatorial optimization. Mathematically, the problem may be stated in the following two equivalent ways:

1- Given a cost matrix D = (d_{ij}), where d_{ij} = cost of going from city *i* to city *j*, (*i*, *j* = 1, 2, ..., n), determine x_{ij} which minimizes the quantity:

$$Q = \sum_{i,j} d_{ij} x_{ij} \qquad (1)$$

Subject to

$x_{ij} = 0$ for $i=j$ (2)

$$x_{ij} = \begin{cases} 1 & \text{If the salesman travels from city } i \text{ to city } j \\ 0 & \text{Otherwise} \end{cases} \qquad (3)$$

$$\sum_i x_{ij} = \sum_j x_{ij} = 1 \qquad (4)$$

and for any subset S ={i_1, i_2, ..., i_r} of the integers from 1 through *n*,

$$x_{i1,i2} + x_{i2,i3} + ... + x_{ir-1,ir} + x_{ir,i1} \begin{cases} < r & \text{for } r < n \\ \leq n & \text{for } r = n \end{cases} \qquad (5)$$

2- Given a cost matrix D as above, find a permutation P, such that

P = (i_1, i_2, i_3, ..., i_n) of integers from 1 through *n* that minimizes the quantity

$$d_{i1,i2} + d_{i2,i3} + ... + d_{in,i1} \qquad (6)$$

Or simply

Minimize $D(d_n, d_1) + \sum_{k=1}^{n-1} D(d_k, d_{k+1})$ (7)

In dealing with problems similar to a large traveling salesman problem, where a really efficient algorithm for the best solution is unavailable, is in general time consuming, if not entirely hopeless, to work on refinement techniques to obtain the best solution. Rather, the approach should be to develop a technique by which good locally optimal solutions can be obtained very fast, and with reasonable probability that among the locally optimal solutions, the best may indeed be found, (in

actual applications, the best of a set of good locally optimal solutions, even though it may not be the best, will be close enough to the best solution as to offer a satisfactory answer for most problems.)

Considering the second way of formulating the problem (Eqs. 6, 7), the nature of the problem looks similar to the FSP discussed above, the application of the FSP heuristics to this problem may produce good solutions, taking into account the reformation of the cost to suit the TSP constraints. Figure-2 shows the computation of the cost function for sequence *s* of size *in.*

As the problem grows larger, the distance matrix increasingly expands. For testing heuristic performance and to avoid the large storage in the computer's limited memory, it is thought that a function for generating these distances should be defined. The test problem is made on cities that lie on the regular square grid which simplify the calculations of distance traveled between any two cities.[7]

```
void cost(in,s,time)
 {
  int s[100];float sum;
  sum=0.0;
  For(j=0,j<in-1,j++)
   {
    y1=(s[j]-1)/10.0;  /* dividing by 10 is
    due to the grid size */
    x1=fmod((s[j]-1),10.0);
    y2=( s[j+1]-1)/10.0 ;
    x2=fmod((s[j+1]-1),10.0) ;
    t=sqrt   (pow((x2-x1),2)   +   pow((y2-
    y1),2)) ;
    sum+=t;
   }
   /* Computation of joining the last city
   with the first one */
   y1=(s[in-1]-1)/10.0;
   x1=fmod((s[in-1]-1),10.0);
   y2=(s[0]-1)/10.0 ;
   x2=fmod((s[0]-1),10.0) ;
   t=sqrt   (pow((x2-x1),2)   +   pow((y2-
  y1),2)) ;
   sum+=t;  time=sum;}
```

Figure-2 Computation of the cost function

4- Genetic Algorithm

4-1- GA Methodology:-

GA are probabilistic search techniques, the probabilistic search procedure in GA, combined with reproduction and recombination, mimic the process of evolution. The fundamental principles of genetics lead to the development of GA [9]. In order to apply GA to a problem, generally the solution space of the problem is represented by a population of structures where each structure is a possible solution to the problem. Then a certain number of structures are chosen to form the initial generation. Applying simple genetic operators to the parent structures selected from the existing generation generates the structures of the next generation.

According to the idea that "*good parents produce better offspring*", a structure with higher fitness value in the current generation will have higher probability of being selected as a parent (similar to the concept of survival). When this process is repeated, a continuous improvement in the structures' performance from one generation to the next can be observed [3]. In most of the applications of GA, the results provided an insight to the robustness of GA in terms of its applicability and quality of the solutions. Figure-3 shows a brief outline of the Genetic Application Approach [3], where:

$S(t)$ is the population in the t^{th} generation;

$s_i(t)$ is the i^{th} member in $S(t)$;

$f(s_i(t))$ is the fitness value of $s_i(t)$,

and $TOTFIT(t)$ is the sum of $f(s_i(t))$ for all $s_i(t)$ in $S(t)$.

Step 1 Generate the initial population, $S(t)$, where $t = 0$. Determine the size of the population, POPSIZE, and the number of generations, GENER.

Step 2 Calculate the fitness value of each member, $f(s_i(t))$, for population, $S(t)$.

Step 3 Calculate the selection probability for each $s_i(t)$, where the selection probability is defined as $P(s_i(t)) = f(s_i(t)) / TOTFIT$.

Step 4 Select a pair of members (parents) that will be used for reproduction via the selection probability.

Step 5 Apply genetic operators (Crossover, Mutation, Inversion) to the parents. Replace the parents with the resulting offspring to form a new population, S(t+1), for generation t+1. If the size of the new population is equal to the POPSIZE, then go to step 6, else go to step 4.

Step 6 If current generation, t+1, is equal to GENER, then stop, else go to step 2.

Figure-3 The Genetic Algorithm Approach

According to the outline of the GA approach, an application of GA should consider the following elements:-

- Representation of Structure,
- Initial Population,
- Population SIZE,
- Selection Probability,
- Genetic Operation, and Termination.

Each of the above elements affects the performance of GA and it is necessary to effectively estimate these fundamental elements.

The GA Application procedure can be divided into two parts:-

1- Implementation of GA to the problem, and

2- Optimization of control parameters of GA.

The first part is mainly concerned with the determination of the representation of structure, the generation of initial population, the approach for determining the selection probability, the genetic operators, and the termination criteria. The second part is concerned with the optimization of the control parameters of GA that will affect the performance of the

implementation of GA. The control parameters usually include the population size, the rates of genetic operators, the generation gap, etc.

The fitness value is calculated as follows:-

1- Calculate the total flow time for each member in the population.

2- Find *Fmax*, which is the maximum total flow time in the population.

3- Calculate the fitness value of each member, which is equal to the difference between the *Fmax* and the total flow time of each member.

4-2- GA Basics:

The most common type of genetic algorithm works like this: a population is created with a group of individuals created randomly. The individuals in the population are then evaluated. The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task. Two individuals are then selected based on their fitness, the higher the fitness, and the higher the chance of being selected. These individuals then "*reproduce*" to create one or more offspring, after which the offspring are mutated randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.[1]

- **Selection**

  While there are many different types of selection, the most common type is the roulette wheel selection. In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness. Two individuals are then chosen randomly based on these probabilities and produce offspring. Pseudo-code for a roulette wheel selection algorithm is given in Figure-4.

```
for all members of population
    sum += fitness of this
individual
end for
for all members of population
    probability = sum of
probabilities + (fitness / sum)
    sum of probabilities +=
probability
end for
loop until new population is full
     do this twice
         number = Random between 0
and 1
       for all members of
population
           if number > probability
but less than next probability
                then you have been
selected
       end for
     end
     create offspring
end loop
```

Figure-4 Pseudo-code for a roulette wheel selection algorithm

- **Crossover**

After selecting the individuals, the most common way to produce offspring with them is *crossover*, and while there are many different kinds of crossover, the most common type is single point crossover. In single point crossover, a locus is chosen at which the remaining alleles can be swapped from one parent to the other. Children take one section of the chromosome from each parent. The point at which the chromosome is broken depends on the randomly selected crossover point. This particular

method is called single point crossover because only one crossover point exists. Sometimes only child 1 or child 2 is created, but oftentimes both offspring are created and put into the new population. Crossover does not always occur, sometimes based on a set probability, no crossover occurs and the parents are copied directly to the new population. The probability of crossover occurring is usually 60% to 70%.

Parent 1	1011010	010100110	**Child1**	1011010	110110101
Parent 2	0011010	110110101	**Child 2**	0011010	010100110

- **Mutation**

After selection and crossover, a new population full of individuals is obtained. Some are directly copied, and others are produced by crossover. In order to ensure that the individuals are not all exactly the same, a small chance of mutation is allowed. all the alleles of all the individuals are looped through, and if that allele is selected for mutation, it can either be changed by a small amount or replaced with a new value. The probability of mutation is usually between 1 and 2 tenths of a percent. Mutation is fairly simple, a change is done to the selected alleles based on what is felt necessary and so on. Mutation is, however, vital to ensuring genetic diversity within the population.

Before
1101101001101110
After
1101100001101110

5- Tests and Results:

Tests involving the application of the heuristics mentioned above resulted in the NEH heuristic proving to be very efficient and accurate up to a 100-city problem, of course, here too the

time involved in such process is somehow excessive, but when compared to other solution methods' results, this heuristic has established very good results specially when combined with the genetic operator schemes.

This heuristic (NEH) starts by constructing an initial route tour for two cities specially selected. The distances from each city to all other cities is summed and stored, then they are reordered in an ascending order, the first two largest sums represent the first partial sequence for the heuristic. It then adds the remaining cities one at a time to form a sequence of partial routes, the last one having N cities in it.

The foregoing heuristic approach focuses on solving a complex situation where the salesman's itinerary includes many cities and alternatives. As each city is added, the computer evaluates all the alternative ways it can be incorporated into the current network. The computer program calculates the effect of inserting the salesman call between each pair of cities that are visited in sequence at the previous stage of analysis. It chooses the most desirable point in the salesman's rout at which to make the call.

As already mentioned, test problems are made on cities that lie on the regular square grid to simplify the calculations of distance traveled between any two cities [7]. For a 100-city problem, NEH heuristic, in about 4.66 second on a Pentium II computer (900 MHz), gave the near optimal solution of 107 units, when improved with genetic improvement procedures, the solution improved to 104 units in about 0.5 second (Figure-5).

The comparison is made with reference to the theoretical exact solution of 101 unit.

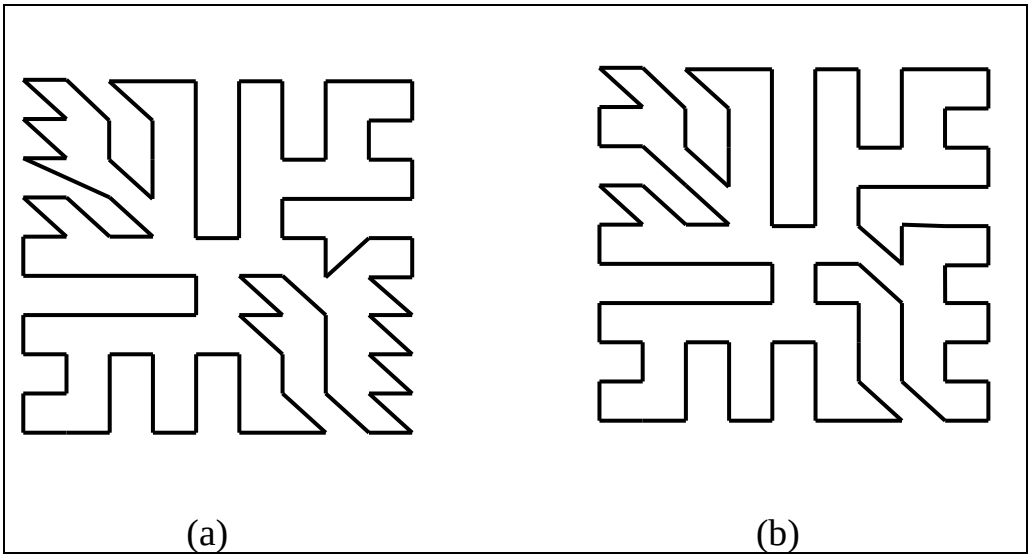

Figure-5 Solutions to a 100-city problem
a- NEH Heuristic solution of 107 unit cost
b- Improved Genetic solution of 104 unit cost

6- Conclusion:

As a concluding remark, it can be seen that a heuristic procedure specific to a problem formulation can be altered somehow to be applicable to solve a different problem and that is only where the problem formulation can be reintroduced to suit the heuristic application. The heuristic procedure itself also has to be modified to account for the difference in some of the characteristics found in the environment of the problem. In this way, a wide range of powerful heuristics can be reused to solve other NP-Complete problems.

The genetic application has also proved to be playing a vital role in improving already obtained solutions towards more optimal solutions in reasonable time limits.

REFERENCES

1. Alex J. Champandard, **Genetic Algorithms Warehouse**, 2001-2002, Artificial Intelligence Depot. At site http://GeneticAlgorithms.ai-depot.com/
2. Cognitive Science 108b Lecture Notes: Heuristic Search at site http://cogsci.ucsd.edu/~batail/108b/lectures/heuristic.html.
3. Cheun-Lung, C., Neppalli, R.V., Aljaber, N.,**"Genetic Algorithms Applied to the continuous Flow Shop Problem**," Computers Industrial Engineering Vol. 30, No. 4, (1996) pp. 919-929.
4. Gendreau M., Hertz A. and Laporte G., **New insertion and post-optimization procedures for the traveling salesman problem**. Operations Research, 1992, 40, 1086-1094.
5. Grtschel M. and Holland O., **Solution of large-scale symmetric traveling salesman problems**. Mathematical Programming, 1991, 51, 141-202.
6. Jean-Louis, L., **"Problem Solving and Artificial Intelligence,"** BPCC Wheatons Ltd, Exeter. 1990 London.
7. Mohammed, A.A**, "Simulating Time-Inhomogeneous Marcov Chains: A Simulated Annealing Optimization Study,"** Ms.C Thesis 1991, University of Mosul.
8. Nawaz, M., Enscore, E.E., Jr., and Ham, I., **"A Heuristic Algorithm for the m-Machine n-Job Flow-Shop Sequencing Problem,"** Omega, 11 (1983), pp. 91-95.
9. Neppalli, V.R., Chuen-lung, C., Gupta, J.N.D., **"Theory and Methodology: Genetic algorithms for the two-stage bicriteria flow shop problem,"** European Journal of Operational Research, Vol. 95 (1996) pp. 356-373.
10. Padberg M. W. and Rinaldi G., **A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problem**. SIAM Review, 1991, 33, 60-100.
11. Renaud J., Boctor F. F. and Laporte G., **A fast composite heuristic for the symmetric traveling salesman problem**. INFORMS Journal on Computing, 1996, 8, 134-143.
12. Zweig G., **An effective tour construction and improvement procedure for the traveling salesman problem**. Operations Research, 1995, 43, 1049-1057.