Solving Traveling Salesman Problem Using Combinational Evolutionary Algorithm

Mohammad Reza Bonyadi[1], S.Mostafa Rahimi Azghadi[1] and Hamed Shah Hosseini[2]
1 Department of Electrical & Computer Engineering, Shahid Beheshti University, Tehran, Iran
{m_bonyadi, m_rahimi} @std.sbu.ac.ir
2 Department of Electrical & Computer Engineering, Shahid Beheshti University, Tehran, Iran
h_shahhosseini@sbu.ac.ir

Abstract. In this paper, we proposed a new method to solve TSP (Traveling Salesman Problem) based on evolutionary algorithms. This method can be used for related problems and we found out the new method can works properly in problems based on permutation. We compare our results by the previous algorithms and show that our algorithm needs less time in comparison with known algorithms and so efficient for such problems.

1 Introduction

It is natural to wonder whether all problems can be solved in polynomial time. The answer is no. For example, there are problems, such as Turing's famous "Halting Problem," that cannot be solved by any computer, no matter how much time is provided. There are also problems that can be solved, but not in time O(   ) for any constant k. Generally, we think of problems that are solvable by polynomial-time algorithms as being tractable, or easy, and problems that require super-polynomial time as being intractable, or hard.
There is an interesting class of problems, called the "NP-complete" problems, whose status is unknown. No polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them. This so-called $P \neq NP$ question has been one of the deepest, most perplexing open research problems in theoretical computer science since it was first posed in 1971[1]. TSP is the problem in this class.

In other hand, in recent years, AI (artificial intelligent) and its searching algorithms becomes attracted. As an example, one approach that has been used in searching problems is Genetic Algorithm to search the space of problems and finding solutions. Nevertheless, these solutions have no guarantee to be the best [6].
Therefore, in this paper we attempt to use a combinational evolutionary algorithm to find the solutions of the TSP and show our algorithm can be used for finding better minimum cycles in the graph, in comparison to preceding algorithms.

2 Background Material

2.1 Traveling Salesman Problem (TSP)

Traveling sales man, is a famous problem that in this problem a person wants to visit all the cities exactly once in his region and back to the first city that ha started his traveling from, assumes that, he wants to minimize his tour value. This problem is a combinational minimization problem and has so many utilizations. The problem been analyzed using many algorithms like branch-and-bound, greedy searching algorithms etc. In recent years, the genetic algorithms been used for analyzing this problem widely [1].
We can assume this problem as an undirected graph problem. In this problem, we are searching for minimum path where visits all the nodes exactly once and finishes at the node start from that. Fig1 indicates an example with its optimal solution. A, B, C… are the cities and the numbers on the edges are the cost of links [1].

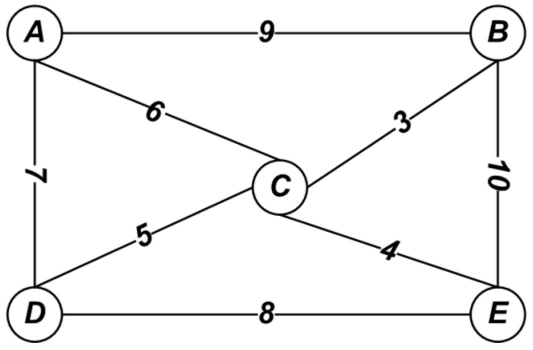

Fig 1 : The tour with A=>B =>C =>E =>D => (A) optimal tour

One way to indicate the solution of the problem is the sequence of the city names. We have to remove the last city from the sequence because it must be same as the first one. Note that for calculating the path value, we do not forget the last city. With indicating the solutions as the permutation of the cities, every city will come exactly once in the sequence. However, some permutations might not be the solution because the graph might not be the complete graph. We can solve this problem by

assuming two nodes that not connected to each other, are connected by the edge that its value is infinite.
We can use a scale function to reach that. As an example consider the whole path value to be S, so we can use the simple function f(S) =S for minimization problem. We can use so many functions like f(S) =1/S or f(S) =1/(S^2) as the functions for maximization [2].
In the condition that our solutions are the permutation of integers, the simple crossover might cause the creating of invalid solutions, too. For example, the one point crossover in 4th place of two follows solutions:

ADEBC, AECDB

Can lead to create these two invalid solutions:

ADEDB, AECBC

We have to use the crossover that always leads to a permutation for solving this problem.

2.2 Partially Matched Crossover

We can use the modified crossover that always leads to a permutation of genes. This modified crossover called partially matched (PMX) crossover. All the problems that their solutions must be the permutations can use this crossover method. We introduce this operation in the following example:
Let

ADEBC, AECDB

be two solutions of the problem and we use the two point crossover in 3rd and 4th indices in the sequence. We call the substrings between crossover points as matching sections. The crossover operation have to change the substrings in the sequences (In this example (EB) from the first sequence and (CD) from second one). Symbol (E, C) shows Replacing E from the first sequence by C. Therefore, the symbol (B, D) shows replacing of fourth index from the first sequence with fourth of the second sequence. The next step of the PMX operation is replacing the elements of these pairs in each sequence. In this example, we have to change the place of E with C and B with D in each sequence. The result of (E, C) in first sequence is ADCBE and the result of (B, D) is ABCDE and the second sequence changes from AECDB to ABCDE and then to ACEBD. As it seems, the results are permutations and have no repetition. Fig 2 can help to understanding the PMX crossover.

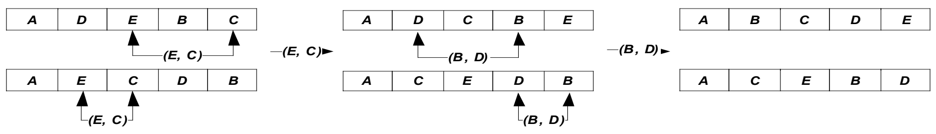

Fig 2: The PMX Crossover

The other GA operations don’t need to change and can be used as the standard GA. Using the GA in combination with the local search algorithms can work better than standard algorithms.

2.3 Introduction to Swap Sequence

We show the swap operator as SO (i1, i2) and define it as follow:
In solution S, we change the place of i1 & i2 and we write as:

S'=S+SO (i1, i2)

As an example, consider S is a solution of a problem and:

S= (1, 3, 5, 2, 4)

So:

S'=S+SO(1,2)=(1,3,5,2,4)+SO(1,2)=(3,1,5,2,4)

2.4 Swap Sequence

The swap sequence is consisting of one or more swap operators.

SS = (SO1, SO2, SO3… SOn)

By applying SS to the solution, SO1 will works first, SO2 will work second and so on.
The different swap sequences might have the same effect on different solutions. We know these sequences as set of equivalent swap sequences. In this set, the basic swap operator is the element that has the minimum swap operators [5].

2.5 Creation of the basic Swap Sequences

Consider we have two solutions named A, B and we want to change B to A using some swap sequences:

SS=A-B -> A=B+SS

Consider:

A= (1, 2, 3, 4, 5), B = (2, 3, 1, 5, 4)

The first swap operator is:

SO1 (1, 3) -> B1 =B+SO1 = (1, 3, 2, 5, 4)

The next one is:

SO2 (2, 3) -> B2 =B1+SO2 = (1, 2, 3, 4, 5)

And the last one is SO (4, 5) and we reach A.
We can define the Move function as follow: with finding difference between two points A, B, we apply some swap operators of swap sequence on B randomly to reach the new solution [5].

3 Proposed algorithm

The proposed algorithm is an evolutionary algorithm where combined from GA idea and Shuffled Frog Leaping (SFL), Civilization and Society algorithms [4].
In each loop, like GA, the elements of production group perform the mutation or crossover in random order. Then for every element of the group, we call a local searching algorithm. Fig 3 indicates its pseudo code [1, 3].

The local search algorithm is the combinational algorithm from SFL and Civilization and Society algorithm. In this phase (Local search), first of all we create the population consisting of P elements. All P elements are same as each other. This is one of the main differences between the proposed algorithm and the SFL algorithm. In SFL, the elements that we perform searching on them are same as the reference set but in proposed algorithm, a population will consist of the same elements and after several loops, the element that has the best fitness, will be replaced by the main element and back to the reference population [4]. Fig.3 shows the local search pseudo code.

```
Pseudo code for a GA procedure
Begin;
 Generate random population of P solutions
 (chromosomes);
 For each individual i∈P: calculate fitness (i);
  For i=1 to number of generations;
   Randomly select an operation
   (crossover or mutation);
   If crossover
    Select two parents at random ia and ib;
    Generate an offspring ic=crossover(ia and ib);
   Else If mutation;
    Select one chromosome I at random;
    Generate an offspring ic=mutate(i);
   End if;
   Local Search(ic);
   Calculate the fitness of the offspring ic;
   If ic is better than the worst chromosome then
    replace the worst chromosome by ic;
  Next i;
 Check if termination = true;
End;
```

Fig 3: Genetic Algorithm pseudo code

Here we will have a glance look at the meaning of memeplexes. According to memetic theory, a **meme** (a unit of cultural information, cultural evolution or diffusion) propagates from one mind to another analogously to the way in which a gene propagates from one organism to another as a unit of genetic information and of biological evolution. Multiple memes may propagate as cooperative groups called *memeplexes* (meme complexes). For more information, see [7].

```
Pseudo code for local Search procedure
local Search(solution S)
Begin;
 Generate a population of P solutions equal to S;
 For each individual i∈P
  Assign a random swap sequence to i;
  calculate fitness (i);
  Divide P into m memeplexes at random;
  For k=1 to number of iterations
   For each memeplexes, set best solution as leader of memeplexes;
   Set best solution of community as community leader;
   For each individual i∈P
    if i is not a leader
     Move i → its group leader
    if i is a group leader
     Move i → the community leader
    End;
  Next k;
End;
```

Fig 4: Local search pseudo code

As an another difference between the proposed algorithm and SFL, we can say that in each loop in SFL, only the worst element in each group will be moved to its group leader. Therefore, if the solution, which implemented by that element, gave better result, the changes will be applied, and in other case, the solution will be moved to the best global solution. If the solution did not change or became worst, we apply a random solution. Nevertheless, in Civilization and Society Algorithm, in each loop, all the elements in similar groups, will move to their group leaders and the leaders will move to the population leader. We use the latter in our proposed approach.
Because the elements, which have been selected by local search, are same as each other, the convergence of population elements differs from the SFL and Civilization and society algorithms. Note that in Traveling Salesman Problem, it is not possible to choose a point as the solution on the line between two known solutions. Because it is not guaranteed that, the new solutions will have any link and relation to their parents. For simulation of moving the solutions close to each other, we use the swap sequence idea.

```
Pseudo code for a SFL procedure
Begin;
  Generate random population of P solutions(frogs);
  For each individual i∈P: calculate fitness (i);
  Sort the population P in descending order of their fitness;
  Divide P into m memeplexes;
  For each memeplexes;
     Determine the best and worst frogs;
     Improve the worst frog position to its best element;
```

```
    Repeat for a specific number of iterations;
  End;
  Combine the evolved memeplexes;
  Sort the population P in descending order of their fitness;
  Check if termination = true;
End;
```

Fig 5: SFL Algorithm pseudo code

```
Pseudo code for Civilization and Society Algorithm
Begin;
  Generate N individuals representing civilization
  Compute fitness;
  Create m clusters based on Euclidean distance
  Identify leader for each cluster
  For each cluster
    For each member I in cluster
      Move I → its leader
    Move leader of cluster → global leader
  End;
End;
```

Fig 6: CS Algorithm pseudo code

4 Experimental Results

The algorithms have tested on three inputs with 30, 89, and 929 points for 50 times. For the first case input, all the algorithms found the optimum solution and we perform our experiments on these algorithms in limited time (60 sec.). As we can see in Table1, in the TSP solved using standard GA with 30 cities, the average time for the implementations was 36 seconds and in 35 times the algorithm found the optimum path. We use our proposed combinational algorithm for solving the problem and the average time for the implementations was 2 seconds. Moreover, in 85% of solving the problems, our algorithm found the optimum path.

Table 1 : The results of 30 point graph for TSP

Algorithm	Time(Sec)	Success percentage
GA	36	70%
GA using SFL method	2	60%
GA using Proposed approach	2	85%

We authorized the algorithm to function in 10 min, then the inputs were 89 points for the first time and 929 points for the second time applied to algorithms and the results are listed in Table 2.
According to the explained algorithms, it seems that each of them have their advantages and disadvantages. Our proposed approach converges faster than the two other algorithms but it seems the local search in complex spaces may not be very efficient and its effects must be reduced proportional to time elapsing.

Table 1

Algorithm	Average path value for 80 point input(million)	Average path value for second input(million)
GA	26	19
GA using SFL method	19	20
GA using Proposed approach	14	18
Exact solution	10	13

References

1. Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest ,"Introduction to Algorithms, " Second Edition Clifford Stein The MIT Press Cambridge , Massachusetts London, England McGraw-Hill Book Company
2. Mitchell Melanie , "An Introduction to Genetic Algorithms, " A Bradford Book The MIT Press, Cambridge, Massachusetts • London, England, fifth printing 1999
3. J.H. Holland, "Adaption in Natural and Artificial Systems, " University of Michigan Press, Ann Arbor (USA), 1975
4. Eusuff, M.M. and Lansey, K.E.,(2003). "Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm." Journal of Water Resources Planning and Management, ASCE, Vol. 129, No. 3, pp. 210-225
5. D.E.Knuth, "The Art of Computer Programming," Vol.3:sorting and Serching. Addison-Wesley, reading, MA, Second Edition, 1998, pp. 222-223
6. Stuart J. Russell and Peter Norvig, "Artificial Intelligence A Modern Approach, " 1995 by Prentice-Hall, Inc.
7. Richard Dawkins, "The Selfish Gene, " 1989 Oxford: Oxford University Press, ISBN 0-19-217773-7 page 192