New binary representation in Genetic Algorithms for solving TSP by mapping permutations to a list of ordered numbers

AMIN MOHEBIFAR
Computer department of faculty of engineering
University of Isfahan
81746-73441 Hezar Jerib st., Isfahan
IRAN

Abstract: - There have been various representations used for encoding TSP tours in Genetic Algorithms. These representations, except binary representation, are forced to define their own crossover and mutation operators, and they cannot work with classical operators. In the other hand, in such cases, there is no guarantee that solutions, which obtained by crossover and mutation operators, are valid. So, we need some operation to repair chromosomes and transform invalid tours to valid tours. This operation is a time-consuming process. However binary representation is not recommended, in this paper we propose our binary representation based on mapping all possible solutions to a list of ordered binary representations which has some advantages.

Key-Words: - Traveling Salesman Problem, Genetic Algorithms, binary representation, vector representation, matrix representation

1 Introduction

Traveling Salesman Problem (TSP) is defined as follows: given a collection of cities (nodes), in order to find the best (minimum cost) tour which visits each city exactly once and then return to starting city. This term, 'traveling salesman', was first used in 1932 [1].

The Traveling Salesman Problem was proved to be NP-hard [2] and therefore any problem belonging to the NP-class can be formulated as a TSP problem. Researchers have suggested many heuristic algorithms, such as genetic algorithms (GAs) [3], for solving TSP [4]. Brady [5] was first researcher who tackled TSP with Genetic Algorithms and his works was followed by others [6] [7] [8].

GAs was introduced by Holland [9]. In GAs the *search space* of one problem is defined as a collection of *individuals* (potential solutions). These individuals are encoded by strings (called *chromosomes*). The purpose of using GAs is to find best individual from the search space. The goodness of an individual is measured with an evaluation function (usually called *fitness function*). In *selection phase*, some individuals are selected. Then, in *crossover phase*, selected individuals are mating together pair by pair and create new individuals (usually called *offspring*). And finally, some of offspring is mutated in *mutation phase*. Each iteration, which consists of selection, crossover and mutation phases, called *generation*. *population* is the set of individuals from the search space which is examined by GAs in each generation.

But before starting GAs for solving a TSP problem, there is one problem: how individuals are encoded and represented? We can divide TSP representation into three categories: Binary Representation, Vector Representation and Matrix Representation. We review these three groups in next sections respectively. In this paper, we introduce our proposed representation that has some advantages. We discuss about it later.

The structure of our paper is as follows: In the following sections (sections 2, 3 and 4), we review exist recommended representations. Then, Section 5 contains our proposed binary representation and its algorithms which are used in GAs. In Section 6 we show comparisons between our representation and others. Last, in Section 7 we provide our conclusions.

2 Binary representation

Common problems, which use GAs, are encoded as sequences of bits. However this approach is not recommended for solving TSP, Lidd [10] describes a GA approach for the TSP with a binary representation and classical operators (crossover and mutation). The illegal tours are evaluated on the basis of complete (not necessarily legal) tours created by a greedy algorithm. The reported results are of surprisingly high quality; however, the largest considered test case consisted of 100 cities only [11].

For example, a tour 1-2-4-3-8-5-0-6-7 is represented:

(0001 0010 0100 0011 1000 0101 0000 0110 0111)

(Memorize mentioned tour, because we represent it in other representations when they are studied)

Michalewicz [11] believes that: There is an agreement in the GA community that the binary representation of tours is not well suited for the TSP, *"It is not hard to see why"*. We discuss about it later that it can be right or not!

3 Vector representation

During the last few years there have been three vector representations considered in connection with the TSP [11]: *adjacency*, *ordinal*, and *path* representations. Each of these representations has its own *genetic operators*.

The *adjacency* representation represents a tour as a list of *c* cities. The city *j* is listed in the position *i* if and only if the tour leads from city *i* to city *j*. In this case, aforesaid tour is represented as follows:

(6 2 4 8 3 0 7 1 5)

The *ordinal* representation represents a tour as a list of c cities; the *i*-th element of the list is a number in the range from 1 to $n-i+1$. The idea behind the ordinal representation is as follows: There is some ordered list of cities *C*, which serves as a reference point for lists in ordinal representations. Assume, for example, that such an ordered list (reference point) is simply *C* = (0 1 2 3 4 5 6 7 8). Here, our example is represented as a list *l* of references, *l* = (2 2 3 2 5 2 1 1 1).

The *path* representation is perhaps the most natural representation of a tour. Our example is represented simply as:

(1 2 4 3 8 5 0 6 7)

4 Matrix representation

There are at least two different representations to construct an evolution program using matrix representation for chromosomes. They are discussed briefly, in this section.

Fox and McMahon [12] represented a tour as a precedence binary matrix M. Matrix element m_{ij} in row *i* and column *j* contains a 1 if and only if the city *i* occurs before city *j* in the tour.

In this representation, the *n.n* matrix M representing a tour (total order of cities) has the following properties:

1. The number of 1s is exactly $\frac{n(n-1)}{2}$.
2. $m_{ii} = 0$ for all $0 \le i \le n$, and
3. if $m_{ij} = 1$ and $m_{jk} = 1$ then $m_{ik} = 1$.

Our example is represented as:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	1	1	0
1	1	0	1	1	1	1	1	1	1
2	1	0	0	1	1	1	1	1	1
3	1	0	0	0	0	1	1	1	1
4	1	0	0	1	0	1	1	1	1
5	1	0	0	0	0	0	1	1	0
6	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	1	1	1	0

The second approach in using matrix representation was described by Seniw [13] and Homaifar and Guan [14] separately. Matrix element m_{ij} in the row *i* and column *j* contains a 1 if and only if the tour goes from city *i* directly to city *j*. This means that there is only one nonzero entry for each row and each column in the matrix (for each city *i* there is exactly one city visited prior to *i*, and exactly one city visited next to *i*).

See our example in this case:

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	1
4	0	0	0	1	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0

However, they used different crossover operators and heuristic inversion.

5 Proposed binary representation

"Unfortunately, there is no practical way to encode a TSP as a binary string that does not have ordering dependencies or to which operators can be applied in a meaningful fashion. Simply crossing strings of cities produces duplicates and omissions. Thus, to solve this problem some variation on standard genetic crossover must be used. The ideal recombination operator should recombine critical

Table 1- ordered tours and their assigned number in decimal and binary formats

Numbers	Binary representation	Tour
0	000	(0 1 2)
1	001	(0 2 1)
2	010	(1 0 2)
3	011	(1 2 0)
4	100	(2 0 1)
5	101	(2 1 0)
6	110	N/A
7	111	N/A

information from the parent structures in a non-destructive, meaningful manner."[15], nevertheless we claim here that it can be wrong!

In proposed method, the some ordered numbers are assigned to all possible solutions. Classical crossover and mutation operators are applied on binary format of these numbers and new calculated binary numbers represents new tours.

Let start with six major questions which is raised when the proposed method is studied:

1- How we assign a number to a tour?
2- How can we find assigned number of a specific tour?
3- How can we find the tour, which a specific number is assigned to?
4- Is it necessary to introduce new crossover or mutation operators?
5- How can we find whether a tour is legal or not?
6- How can we repair an illegal tour?

Each answer is divided into 2 sections: first section is an example (a *three*-city problem); next section is in general (a *c*-city problem).

Answer of question 1: suppose that we have a *three*-city TSP problem with cities, which are labelled 0, 1, and 2. The number of all possible solutions are $3!$ (In other hand, the search space has 6 members). We can assign the number 0 to 5 to these 6 tours as follows:

Just like alphabetic ordering that a word starting with *'a'* is before a word starting with *'b'* and so on, we suppose that a tour starting with city 0 is before a tour starting with city 1 and so on. And, if two tours start with same city, for example (1 0 2) and (1 2 0), then we must compare next cities of them (So, 1-0-2 is prior to 1-2-0). Table 1 shows all 6 tours and their assigned number (in decimal and binary formats).

We know that for representing these 6 tour, we need $\lceil \lg(6) \rceil$ or 3 bits. Note that there are no tours which the numbers 6 and 7 are assigned to them. These numbers are illegal and if they are created as new individuals (by crossover and mutation), they must be repaired (see answer of questions 4 and 5 for more explanations).

In general, for a *c*-city TSP problem (cities are labelled from 0 to *c*-1), the number of all potential solutions (all permutations) is equal to $c!$. For sorting these permutations, we assume that:

$$\text{city } i < \text{city } i+1 \text{ ; for each } 0 < i < c$$

Hence, the tours starting with *i* are before tours starting with *i*+1 and if j-th ($0 < j <$ c) city of two tours are same, we compare *j*+1-th city of them and so on.

For representing these $c!$ tours in binary representation, we can assign numbers 0 to $c! - 1$ to them. So, we need b=$\lceil \lg(c!) \rceil$ bits. Note that the numbers $c!$ to $2^b - 1$ are illegal, and they must be transformed to legal tours.

Answer of question 2: Suppose we want to find the number assigned to tour (1 0 2). Firstly, like *ordinal representation*, we create an ordered list *l*, which consists of labels of all cities (note that indices of cities in *l* start from 0). Hence,

$$l = \{0, 1, 2\}$$

First city in supposed tour is 1. This city is second city in *l* (its index is 1). The assigned number to this tour is greater than or equal to $1*(3-1)!$. Now we remove 1 from *l*. Thus:

$$l = \{0, 2\}$$

Next number in supposed tour is 0. It is *0*-th number in new *l*. In this step, the assigned number to this tour is greater than or equal to $1*(3-1)! + 0*(3-2)!$. If we remove 0 from *l*, the number of its members becomes 1 and calculation is stopped. Desired number is equal to last obtained number (i.e. $1*(3-1)! + 0*(3-2)!$ or 2).

In general, the assigned number *n* of a tour *t* in a *c*-city (labelled by numbers 0 to *c*-1) TSP problem is obtained as follows:

```
n = 0
l = {0, 1, 2..., c-1}
for (i=1; i<c; i++){
        x=t[i-1]
        ind= l.indexof(x)
        n+= ind*$(c-i)!$
        l.remove(x);
}
```

After finishing above pseudo-code *n* is assigned number of *t*.

Table 2- all steps for calculating *n* which is assigned to tour 1-2-4-3-8-5-0-6-7 according to proposed algorithm shown in answer of question 2. Before starting these steps, $l = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ and n=0 (in this case, c =9)

i	*l (before removing x)*	*x*	*ind*	*ind**$(c-i)!$	*n*	*l (after removing x)*
1	{0,1,2,3,4,5,6,7,8}	1	1	1 * (9-1)! = 40320	40320	{0,2,3,4,5,6,7,8}
2	{0,2,3,4,5,6,7,8}	2	1	1 * (9-2)! = 5040	45360	{0,3,4,5,6,7,8}
3	{0,3,4,5,6,7,8}	4	2	2 * (9-3)! = 1440	46800	{0,3,5,6,7,8}
4	{0,3,5,6,7,8}	3	1	1 * (9-4)! = 120	46920	{0,5,6,7,8}
5	{0,5,6,7,8}	8	4	4 * (9-5)! = 96	47016	{0,5,6,7}
6	{0,5,6,7}	5	1	1 * (9-6)! = 6	47022	{0,6,7}
7	{0,6,7}	0	0	0 * (9-7)! = 0	47022	{6,7}
8	{6,7}	6	0	0 * (9-8)! = 0	47022	{7}

Table 2 shows all steps for obtaining number for our aforesaid example: 1-2-4-3-8-5-0-6-7. As shown, *n* is 47022 and it means that in proposed ordered list of all permutations, the rank of this tour is 47022. The number of bits we need is $\lceil \lg(9!) \rceil$ (i.e. 19) and this tour is represented as:

0001011011110101110

Compare this length (19 bits) with length of representing with binary or vector representations (36 bits) and matrix representations (81 bits). We discuss about it in section 6 in detail.

Answer of question 3: Suppose we want to find a tour which number 4 is assigned to it. Like previous answer, let $l = \{0, 1, 2\}$. 4 module (3-1)! = 2, so first city is *2*-nd city in *l* (i.e. city 2). We remove it from *l* and repeat algorithms by remainder of 4/(3-1)! (i.e. 0). 0 module (3-2)! = 0, so next city is *0*-th city in *l* (i.e. city 0). The algorithm ends because *l* has one member and it is last city in tour.

In general, in a *c*-city (labelled by numbers 0 to *c*-1) TSP problem the tour *t*, which a number *n* is assigned to it, is obtained as follows:

```
l = {0, 1, 2..., c-1}
t=()
for (i=1; i<=c; i++){
        q= n / $(c-i)!$
        r= n % $(c-i)!$
        x=l.elementAt(q)
        l.remove(x)
        t.add(x)
        n=r
}
```

After finishing above pseudo-code, desired tour *t* is constructed.

Table 3 shows all steps for obtaining the tour in a 9-city TSP problem when its assigned number *n* is 47022. As shown, the tour is our example tour which its number was calculated in previous answer.

Answer of question 4: Absolutely not! Classical operators (crossover and mutation), which can manipulate any bit strings, are enough. They are proposed by Holland [9].

Answer of question 5: Just by using an if statement:

```
if !(n<c!) then
        tour is illegal
```

Answer of question 6: There are some ways to repair illegal tours:

1- Set last bit (MSB) to 0: Most Significant Bit of an illegal tour is 1 certainly. So, if we set MSB to 0, an illegal tour becomes legal.
2- Complement all of bits: like above, MSB is 1 and by complementing, it becomes 0.
3- Subtract population size from it: new tour is in legal range.

By applying these three ways exactly once, we can obtain legal tours.

6 Comparison

The first comparison case is memory comparison between four approaches (See table 4). As we know:

$$\theta(n!) < \theta(n^n) < \theta(2^{n^2})$$

Thus,

$$\lceil \lg(n!) \rceil < n.\lceil \lg(n) \rceil < n^2$$

Table 3- all steps for creating *t* when *n* is 47022 (in this case, *c* =9)

i	*l (before removing x)*	*n*	$(c-i)!$	*q*	*r*	*x*	*l (after removing x)*	*t*
1	{0,1,2,3,4,5,6,7,8}	47022	8! = 40320	1	6702	1	{0,2,3,4,5,6,7,8}	1
2	{0,2,3,4,5,6,7,8}	6702	7! = 5040	1	1662	2	{0,3,4,5,6,7,8}	1-2
3	{0,3,4,5,6,7,8}	1662	6! = 720	2	222	4	{0,3,5,6,7,8}	1-2-4
4	{0,3,5,6,7,8}	222	5! = 120	1	102	3	{0,5,6,7,8}	1-2-4-3
5	{0,5,6,7,8}	102	4! = 24	4	6	8	{0,5,6,7}	1-2-4-3-8
6	{0,5,6,7}	6	3! = 6	1	0	5	{0,6,7}	1-2-4-3-8-5
7	{0,6,7}	0	2! = 2	0	0	0	{6,7}	1-2-4-3-8-5-0
8	{6,7}	0	1! = 1	0	0	6	{7}	1-2-4-3-8-5-0-6
9	{7}	0	0! = 1	0	0	7	{}	1-2-4-3-8-5-0-6-7

Table 4- comparison between four representation approaches

Approach	Memory (bits)
Binary (Lidd)	$n.\lceil \lg(n) \rceil$
Vector	$n.\lceil \lg(n) \rceil$
Matrix	n^2
Proposed binary	$\lceil \lg(n!) \rceil$

In addition, for proposed binary representation, time complexity of creating new individuals, finding illegal offspring and repairing illegal tours are $\theta(1)$. Unfortunately, no results about time complexity are found for others representations, but it is obvious their time complexity is worse than $\theta(1)$ (except for ordinal representation that no illegal tours are created and it does not need any extra operation).

7 Conclusion

Needless to say, that proposed method for TSP tour representation are much better than others in time and space. However, in this paper statistical topics are avoided, it seems that reasons are reasonable.

For future works, two major topics are focused:

1. Comparison of time complexity of creating new individuals, finding illegal offspring and repairing illegal tours by using statistics.
2. Testing some benchmarks to find out whether we can obtain better solutions or not (because we use classical operators, we can guess GAs can explore more room of search space and better solutions are obtained).

References:

[1] E.L. Lawer, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *The Traveling Salesman Problem*, Chichester, UK : John Wiley, 1985

[2] M. Garey, and D. Johnson, *Computers and Intractability*, San Francisco: W.H. Freeman, 1979

[3] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*, New York: Addison-Wesley, 1989

[4] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, *Artificial Intelligence Review,* Vol. 13, 1999, pp. 129-170.

[5] R.M. Brady, Optimization Strategies Gleaned From Biological Evolution, *Nature*, Vol. 317, 1985, pp. 804 – 806.

[6] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, , Genetic Algorithms for the TSP. *Proc. 1st International Conf. on Genetic Algorithms and Their Applications*, Hillsdale, New Jersey, 1985, pp. 160-165.

[7] D.E. Goldberg, and Jr.R. Lingle, Alleles, Loci and, the TSP, *Proc. 1st International Conf. on Genetic Algorithms and Their Applications*, Hillsdale, New Jersey, 1985, pp. 154 – 159.

[8] I.M. Oliver, D.J. Smith and J.R.C. Holland, A Study of Permutation Crossover Operators on the TSP. *Proc. 2nd International Conf. on Genetic Algorithms and Their Applications*, Hillsdale, New Jersey, 1987, pp. 224-230.

[9] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan, 1975

[10] M.L. Lidd, *Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms*, Technical Report, MITRE Corporation, 1991

[11] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1996

[12] B.R. Fox, and M.B. McMahon, Genetic Operators for Sequencing Problems. *1st Workshop on the Foundations of Genetic Algorithms and Classifier Systems*, San Mateo, CA, 1991, pp. 284-300.

[13] D.A. Seniw, *Genetic Algorithm for the Traveling Salesman Problem*, MSc Thesis, University of North Carolina at Charlotte, 1991.

[14] A. Homaifar, & S. Guan, *A New Approach on the Traveling Salesman Problem by Genetic Algorithm*, Technical Report, North Carolina A & T State University, 1991.

[15] D. Whitley, T. Starkweather, & D'A. Fuquay, Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator, *Proc. 3rd International Conf. on Genetic Algorithms*, San Mateo, CA, 1989, pp. 133-140.