IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 12, Issue 4 Ver. IV (Jul. - Aug.2016), PP 82-86
www.iosrjournals.org

An electronic algorithm to find the optimal solution for the travelling salesman problem

M. Sghiar
9 Allée capitaine J.B. Bossu, 21240, Talant, France.

Abstract: *I give here an electronic algorithm of the order* $O(n^3)$ *which is a generalization of the atomic algorithm found in [6] and allows us to find the optimal Hamiltonian cycles.*
If the atomic algorithm (see [6] and [5]) was inspired by the movement of the particles in the atom, the electronic algorithm is inspired by the resistor in the electrical circuit.
Keywords*: Graph, Hamilton cycles, P=NP, the travelling salesman problem, TSP.*

I-Introduction :

The travelling salesman problem (TSP), which is an NP-hard problem in combinatorial optimization (see : [1] , [2] , [3] and [4]) important in operations research and theoretical computer science, asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

In the articles [5] and [6], I was interested to find the Hamiltonian cycles in a graph - not necessarily optimal-. And inspired by the movement of the particles in the atom, I demonstrated (in [5] and [6]) the existence of a polynomial algorithm of the order $O(n^3)$ for finding Hamiltonian cycles in a graph.

This algorithm I called Atomic algorithm (See [6]), joins other several methods to find the Hamiltonian cycles like the Monte Carlo method, Dynamic programming, or DNA computing. And to prevent memory overflow and the slow execution of the program, I suggested in [6] the use of servers: Each point x_i in the graph will be considered as a server, and each server x_i will communicate with each other server x_j with which it is connected . And finally the server x_0 will receive and display the Hamiltonian cycles if they exist.

But I found that the full of memory is caused by the fact that each server send all the list he received, hence the idea to limit the amount that will send each server, and delete as and as the list that do not serve for anything. This idea was great, since I no longer have the problem of memory overflow and the program runs faster : thus only in ten minutes, with a non- powerful computer, I was able to find a Hamiltonian cycle for thousand cities: You can test it with the below improved Feynman code.

And to find an optimal solution to the TSP problem, the solution to find all Hamiltonian cycles and choose the cycle that have the shortest distance would take time hugely although servers operate simultaneously. Hence the need to seek other way to find the optimal solutions. Thinking of the servers, we immediately think of antennas, microwave , frequencies .... in short, we think to the field of electronics. Hence the idea: If we saw servers or nodes of the graph as a node in an electrical circuit, and the "distances" between vertices as a resistor values in a circuit, then more the resistor's value is low and more the current passes quickly, so the particle or its energy will move faster by taking the shortest route.

II- Improved of the Feynman code that gives a Hamiltonian cycles :

```
from scipy import *
import numpy as np
import random
import                                                    time
```

```
start = time.time()
# The Feynman code in Python:
# Written by sghiar July 21, 2016 at 21:25 p.m..
# This code allows you to find the  Hamilton cycle if it exists.
# Skip Code
# We define the Feynman function F.
def F(j, T):
   l= len(T)
   U=[l+1]
   U=[0]*(l+1)
   U[0]=T[0]-1
   for i in range(1,l):
      U[i+1]=T[i]
   U[1]=j
   return U

# We define the function R.
def R(T):
        l= len(T)
        U=[]
        for i in range(l-1):
                U.append(T[i+1])
        return U

# We define the distance function in a Hamiltonian cycle.

def D(T):
   D=0.0
   l= len(T)
   for i in range(0,l-1):
      D=D+(G[T[i]][T[i+1]])
   return D

# We construct the graph G :

print ("number of cities=")

n=input()

G=np.eye(n,n) #

for i in range(n):
   for j in range(n):
      G[i][j]=1
      #G[i][j]=input()
      #print "G[",i,"][",j,"]"
      #G[i][j]=input()
                #if i<=j :
                        #G[i][j]=random.randint(0,1)
                #else:    G[i][j]=G[j][i]
                #print "G[",i,"][",j,"]=",G[i][j]

d={}

d[0]=[[n,0]]

for j in range(n):
      if G[0][j]!=0 and 0!=j :
         d[j]=[[n-1,j,0]]
```

```
            d[0]=[]
            #print d[j]
        else :
                    d[j]=[[0,j]]
                    #print d[j]


L=[]
H=[]

for k in range(0,n**2) :

    if  len(H) != 0 :

            print H
            print("Time:", time.time() - start)
            break

    print(k, "Time:", time.time() - start)
    print "The program is looking for the Hamiltonien cycles..."

    if k%n==0:

        for T in d[k%n]  :
            if  T[0] == 0 :
                H.append(T)

            else:
                pass

        del d[0]
        d[0]=[]


    elif k%n!=0 and len(d[k%n])>0:
                                l=len(d[k%n])
                                T=d[k%n][l-1]
                                for j in range(0,n):

                                        if T[0]<=0 or (j in R(T) and j!=0):
                                                pass
                                        else :
                                                if  G[k%n][j]!=0 and (k%n)!=j  :
                                                        d[j]+=[F(j,T)]
                                                else:
                                                        pass
                                d[k%n].remove(T)


#Hamiltonians Cycles  :

if len(H)!=0:
            for elt in H:

                                print ("There exist the Hamiltonian cycles")
                                print(R(elt) ," Is one Hamiltonien cycle, Its distance is :" , D(elt) )

else :
    print("No Hamiltonian cycles ")
```

print len(H)
print("Time:", time.time() - start)
 # End of code

III- An electronic algorithm to find the optimal solution for the travelling salesman problem.

This algorithm is almost similar to the Feynman algorithm (See [5]) to a slight change near : Let G be a graph on $E=\{x_0,....,x_{n-1}\}$, where $G[x_i][x_j]=1$ or 0, and the "distance" between x_i and x_j is noted d_{ij} .

we can reduce to the case where d_{ij} is a nonzero positive : $d_{ij}>0\,,\forall i,j$

We note $\{d_0,....,d_l\}=\{d_{ij}\}$ with $d_i\leq d_j$ if $i\leq j$.

As we have seen in the introduction, thinking of the servers, we immediately think of antennas, microwave , frequencie .... in short, we think to the field of electronics. Hence the idea: If we saw servers or nodes of the graph as a node in an electrical circuit, and the "distances" between vertices as a resistor values in a circuit, then more the resistor's value is low and more the current passes quickly, so the particle or its energy will move faster by taking the shortest route.

And there is a link between time and the "distance" :

$$t_{ij}=\frac{1}{v_{ij}}d_{ij}$$

Where v_{ij} is the speed of the electrons in the electrical circuit.

So :

$$\omega_{ij}=v_{ij}\frac{1}{d_{ij}}$$

Where ω_{ij} is the frequency.

Otherwise seen : The server i communicate with the server j with the frequency ω_{ij} .

Time progresses, the server x_0 sends the energy to other servers, the other servers will send it in turn to others. But as the exchange between two servers x_i and x_j occurs at a frequency ω_{ij} , to a factor, let $v_{ij}=1$.

We shall have :

$$t_{ij}=d_{ij} \text{ and } \omega_{ij}=\frac{1}{d_{ij}}$$

We start with the **high frequencies** : in other words by the smallest resistor value.

We first see the smallest resistor value d_{s_0} between x_0 and x_i such as $G[x_0][x_i]=1$.

In the step i=1 (mod l) :

x_0 will give its energy to the other points x_i through the resistors d_{s0} .

And as in the Feynman algorithm, we construct the Feynman vector for the point x_i :

$$F(x_i)=\begin{pmatrix} E_{x_i} \\ x_i \\ \vdots \\ x_0 \end{pmatrix}$$

And we construct the Feynman matrix for the point x_i :

$$M(F,x_i)=\begin{pmatrix} E_{x_i} & \cdots \\ x_i & \cdots \\ \vdots & \cdots \\ x_0 & \cdots \end{pmatrix}$$

In this step, those points x_i will have as energy value $E_{x_i}=n-1$.

In the step i=2 (mod l) :

Among the points x_i related to x_0 we choose the minimum resistor value d_{s1} such that $d_{s1}>d_{s0}$ if it exist, else we choose the minimum resistor value d_{s1} such that $d_{s1}\leq d_{s0}$.

Each point x_i linked to a resistor with value d_{s1} will give its energy to other points through the resistor d_{s1} .

In the step i+1 (mod l) :

We continue this process and as the energy decreases; the algorithm, as the Feynman algorithm, will stop, and will display the Hamiltonian cycle with a shorter distance (if the Hamiltonian cycles exist).

Example :

We can check this technique for example for the Graph G on the basis with : G(x, y)=1, , and with the distances : d(a,c)=3, d(a,b)=d(a, d)=d(b,d)=1, d(b, c)=2, d(c,d)=4. We check that acbda is the first hamiltonian cycle that we find and that it have a minimal distance 7. The other cycle with the distance 8 will be found after because it do not have a minimum distance, and the cycle with the distance 9 will not be found for the same reason.

Note :

1- The above improved Feynman algorithm indicates whether the Hamiltonian cycles exist.

2- If the Hamiltonian cycles exist, the electronic algorithm allows us to find the optimal Hamiltonian cycles.

3- The electronic algorithm is also of the order $O(n^3)$ like the Feynman algorithm.

4- The electronic algorithm is a generalization of the Atomic algorithm.

5- With this way I could find by hand the optimal Hamiltonian cycles for some graphs of a small size : See the example above.

References

[1] Lizhi Du. A polynomial time algorithm for hamilton cycle. IMECS, I:17–19, March 2010. Hong Kong.

[2] L.Lovasz. Combinatorial problems and exercises. Noth-Holland, Amsterdam, 1979.

[3] D.S.Johnson M.R.Garey. Computers and intractability:a guid to the theory of np-completeness. Freeman,San Francisco, 1979.

[4] R.Diestel. Graph theory. Springer, New York, 2000.

[5] M. Sghiar. Algorithmes quantiques, cycles hamiltoniens et la k-coloration des graphes. Pioneer Journal of Mathematics and Mathematical Sciences, 17-Issue 1:51–69, May 2016.

[6] M. Sghiar. Atomic algorithm and the servers' s use to find the hamiltonian cycles. International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, 6-Issue 6:23–30, jun 2016.