

## Atomic algorithm and the servers' s use to find the Hamiltonian cycles

M. Sghiar

UFR de Mathématique et d'Informatique / 7, rue René Descartes - 67084 Strasbourg Cedex France

### ABSTRACT:

Inspired by the movement of the particles in the atom, I demonstrated in [5] the existence of a polynomial algorithm of the order  $O(n^3)$  for finding Hamiltonian cycles in a graph with basis  $E = \{x_0, \dots, x_{n-1}\}$ . In this article I will give an improvement in space and in time of the algorithm says: we know that there exist several methods to find the Hamiltonian cycles such as the Monte Carlo method, Dynamic programming, or DNA computing. Unfortunately they are either expensive or slow to execute it. Hence the idea to use multiple servers to solve this problem : 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.

**Keywords :** Graph, Hamilton cycles, P=NP

### I. Introduction

It is known both theoretically and computationally so difficult to find a Hamilton cycles(paths) in simple graphs, and that this problem is a classical NP Complete problem. ([1], [2], [3] and [4]).

Inspired by the movement of the particles in the atom, I demonstrated in [5] the existence of a polynomial algorithm of the order  $O(n^3)$  for finding Hamiltonian cycles in a graph. In this article I will give an improvement in space and in time of the algorithm says:

we know that there exist several methods to find the Hamiltonian cycles like the Monte Carlo method, Dynamic programming, or DNA computing. Unfortunately they are either expensive or slow to execute it. Hence the idea to use multiple servers to solve this problem.

The first code is faster but consumes more memory, while the second, although slower, but uses less memory since it uses writing to files during the search of Hamilton cycles.

These two algorithms inspired the idea of the servers' use.

### The First code :

```
from scipy import *  
import numpy as np  
import random  
import time  
start = time.time()
```

```
# The Feynman code in Python:  
# Written by sghiar 24 may, 2016 was 21:25 p.m..  
# This code allows you to find the Hamilton cycle if it exists.
```

```
# Skip Code
```

```
# We define the function F Feynman.
```

```
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 Hamiltonian 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:
        for T in d[k%n] :

            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
```

```
del d[k%n]
d[k%n]=[]

#Hamiltonians Cycles :

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

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

    else :
        print("No Hamiltonian cycles ")

# End of code
```

#### The second code :

And if we want to use less memory ram , we can use this algorithm:

```
/usr/bin/env python
#coding=utf-8
import decimal
from scipy import *
import numpy as np
import random
import time
start = time.time()
import os

# The Feynman code in Python:
# Written by sghiar 28 may, 2016 may 14:53 p.m..
# This code allows you to find the Hamiltonian cycle if it exists.

# Skip Code

# We define the function F Feynman.

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

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[l]=j
return U

# We define the distance function in a Hamilton 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={}

f={}
for i in range(n):
    f[i]= file( "fichier_%.d.txt"%i, "w")

f[0].write(str([n,0])+"\n")

for j in range(1,n):
    if G[0][j]!=0 and 0!=j :
        f[j]= file( "fichier_%.d.txt"%j, "a")
        f[j].write(str([n-1,j,0])+"\n")
```

```
else :  
  
    f[j].write(str([0,j])+"\n")  
  
L=[]  
H=[]  
  
for k in range(0,n**2) :  
  
    if len(H) != 0 :  
  
        print H  
        print("Time:", time.time() - start)  
        break  
  
    else:  
  
        print(k, "Time:", time.time() - start)  
        print "The program is looking for the Hamiltonian cycles..."  
  
    if k%n==0:  
  
        #f[k%n]= file( "fichier_%d.txt"%(k%n), "r")  
        f[k%n]= open( "fichier_%d.txt"%(k%n), "r")  
  
        for T in f[k%n] :  
            exec("T='+T)  
            x=T  
  
            if x[0] == 0 :  
  
                H.append(R(F(j,T)))  
                break  
  
            else:  
                pass  
  
        f[0].close()  
        del f[0]  
        f[0]= file( "fichier_%d.txt"%(k%n), "a")  
        #os.remove("fichier_0.txt")  
  
    elif k%n!=0:  
  
        f[k%n]= open( "fichier_%d.txt"%(k%n), "r")  
  
        for T in f[k%n]:  
  
            T.split('=')  
            exec("T='+T)
```

```
x=T
for j in range(0,n):
    if x[0]<=0 or (j in R(x) and j!=0):
        pass
    else :
        if G[k%n][j]!=0 and (k%n)!=j :
            f[j]= file("fichier_%d.txt"%j, "a")
            f[j].write(str(F(j,x))+"\n")
            f[j].close()
        else:
            pass

del f[k%n]
#os.remove("fichier_%d.txt"%(k%n))
f[k%n]= file( "fichier_%d.txt"%(k%n), "a")

#Hamiltonians Cycles :

if len(H)!=0:
    for elt in H:
        print ("There exist the Hamiltonian cycles")
        print(R(elt) ," Is one Hamiltonian cycle, Its distance is :", D(elt) )
    else :
        print("No Hamiltonian cycles ")

for i in range(n):
    f[i].close()

# End of code
```

**Note: The two algorithms above can be modified to use at least n servers to find the Hamiltonian cycles, so we will win time and space ( memory):**

### The third code :

There exist several methods to find the Hamiltonian cycles such as the Monte Carlo method, Dynamic programming, or DNA computing. Unfortunately they are either expensive or slow to execute it. Hence the idea to use multiple servers to solve this problem.

Each point  $x_i$  in the graph will be considered as a server, and each server  $x_i$  will send  $F(j,T)$  -T is in  $d[i]$ - to

each server  $x_j$  with which it is connected . And finally the server  $x_0$  will receive and display the Hamiltonian cycles if they exist.

Obviously the servers can work simultaneously, which speeds up the execution of the program and solves the problem of the full memory.

## 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.