2013年度 卒業論文

ログイン情報を用いた
ソーシャルグラフの作成

大阪産業大学 工学部 情報システム工学科
情報教育システム研究室

10H018 奥田侑紀

目次

1　はじめに

学生の交友関係は目で視ることができない。視ることができないということは、交友関係の把握や他人との認識の差異、認識の共有など不便な点がある。そこで、プログラミング演習室 (以下、演習室という) のログイン情報からソーシャルグラフを作成することで、学生の交友関係の可視化を試みた。本研究では、大阪産業大学 (以下、本学という) デザイン工学部情報システム学科の学生を対象とし、演習室のログイン情報を用いて学生の交友関係の可視化を行い、ソーシャルグラフで学生の交友関係が把握できるかを検証した。

第 2 章では本研究の目的を説明する。第 3 章では本研究で使用する用語について説明する。第 4 章では、本研究で制作したシステムを説明する。第 5 章では、本研究を考察する。第 6 章では、本研究の課題について説明する。

2　本研究の目的

本研究の目的は、学生の交友関係をソーシャルグラフとして可視化することである。

演習室のコンピュータのログイン情報からソーシャルグラフを作成し、学生の交友関係を可視化する。本学のデザイン工学部情報システム学科は講義で演習室を使用することが多い。演習室の端末ログイン情報から着席位置を割り出し、交友関係を把握するシステムを開発した。

3 用語解説

この章では, 本研究に用いた用語の解説を行う。

3.1 グラフ

本研究で扱うグラフは「グラフ理論[*1]」のグラフのことである。グラフ理論のグラフは、ノード[*2]とエッジ[*3]の集合体である。ノード同士の関係や経路などを表すグラフである。

3.2 ソーシャルグラフ

ソーシャルグラフとは、Mr.Brad Fitzpatrick [1] によって提唱された人間同士の相関のことを指す概念のことである。ソーシャルグラフと言われる多くはグラフのことを指される。ソーシャルグラフでは、ノードが人、エッジが人と人の関係性であり、人間の相関図を表す。Facebook や Twitter や SNS などのソーシャルグラフが多く作成されている [2,3]。

[*1] 数学の一分野である。グラフの性質を研究する学問のことである。
[*2] 要素を表す円のこと。
[*3] ノード同士の関係を表す線のこと。

4　ソーシャルグラフ作成システム

4.1　システム概要

本研究では、ログイン情報から交友関係を判断しソーシャルグラフを作成するシステムを開発した。演習室の端末のログイン情報から学生の着席状況を判断し、本学デザイン工学部情報工学科の交友関係を表すソーシャルグラフを出力する。システムは Pyhton で作成し、ソーシャルグラフの計算には Networkx [4] を使用した。

4.2　システムの動作

以下にシステムの動作を示す。

1. 端末のログイン情報を取得する。
2. 学生ごとに、誰と隣に着席していた時間を算出し、集計する。
3. ログインした学生をノードとする。集計された時間が平均以上の学生同士に対してのみ、エッジを引く。
4. ソーシャルグラフを見やすいように変更する。
5. ソーシャルグラフを出力する。

4.3　ログイン情報

本研究では交友関係を求める元データとして、演習室の端末ログイン情報を使用する。ログイン情報は rwho により、学生がログインをしているか判断し取得する。ログイン情報の例を **図 1** に示す。演習室の端末のログイン情報を使用する理由は、本学デザイン工学部情報工学科は必修の講義を含め演習室を多く利用するため多くのデータが取れるからである。端末のログイン情報から学生の着席している席を判断する。1 人学生を対象に取り、同時刻に対象学生の隣に着席している学生と隣に着席していた時間を集計していく。偶然隣に着席して場合、集計されていく時間は増えない。友達ならば隣に着席するため、他人とは比べると明確な差が出るほど時間は増えていく。それにより、他人か友人かを判断できると考えた。

```
s51h999@ES63:win0 2013/04/15 10:27 - 2013/04/15 11:24
s54h700@WS39:win0 2013/04/15 10:30 - 2013/04/15 11:27
s52h310@WS69:win0 2013/04/15 10:30 - 2013/04/15 11:27
s52h516@ES71:win0 2013/04/15 10:30 - 2013/04/15 11:27
s51h813@WS64:win0 2013/04/15 10:33 - 2013/04/15 11:30
s51h965@WS41:win0 2013/04/15 10:36 - 2013/04/15 11:33
s51h867@WS75:win0 2013/04/15 10:37 - 2013/04/15 11:34
s52h697@ES18:win0 2013/04/15 10:36 - 2013/04/15 11:34
s53h548@WS71:win0 2013/04/15 10:37 - 2013/04/15 11:34
s53h406@WS22:win0 2013/04/15 10:38 - 2013/04/15 11:35
s53h237@WS49:win0 2013/04/15 10:41 - 2013/04/15 11:35
s53h341@WS27:win0 2013/04/15 10:44 - 2013/04/15 11:35
s53h570@WS36:win0 2013/04/15 10:38 - 2013/04/15 11:35
s53h478@WS24:win0 2013/04/15 10:38 - 2013/04/15 11:35
```

図 1　使用するログイン情報の例。ログイン名、端末番号、エントリー、ログインとログアウトの日時と表示される。

4.4 パラメータ

演習室の端末のログイン情報から、学生がいつ、どこに着席していたかを判断する。ログインしていた学生の隣に他の学生が着席した場合、隣り合っていた時間を 2 名の学生のエッジに格納する。指定期間内にログインした全学生に同じ処理を行う。そして、全エッジを対象に格納された時間の平均を出す。平均のままのパラメータの場合、友達ではないものエッジを含めているために平均は低いものになる。エッジの重みが平均以上のものをグラフに書き出す。

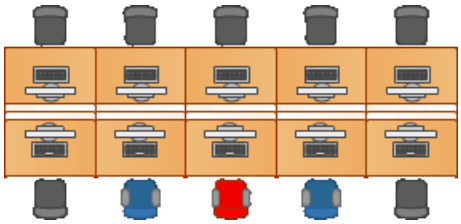

図 2　赤色の席を対象学生とする場合、青色の席に着席している学生と隣同士に着席した時間を集計する。

例として、エッジに対しての時間の分布を**図 3** に示す。

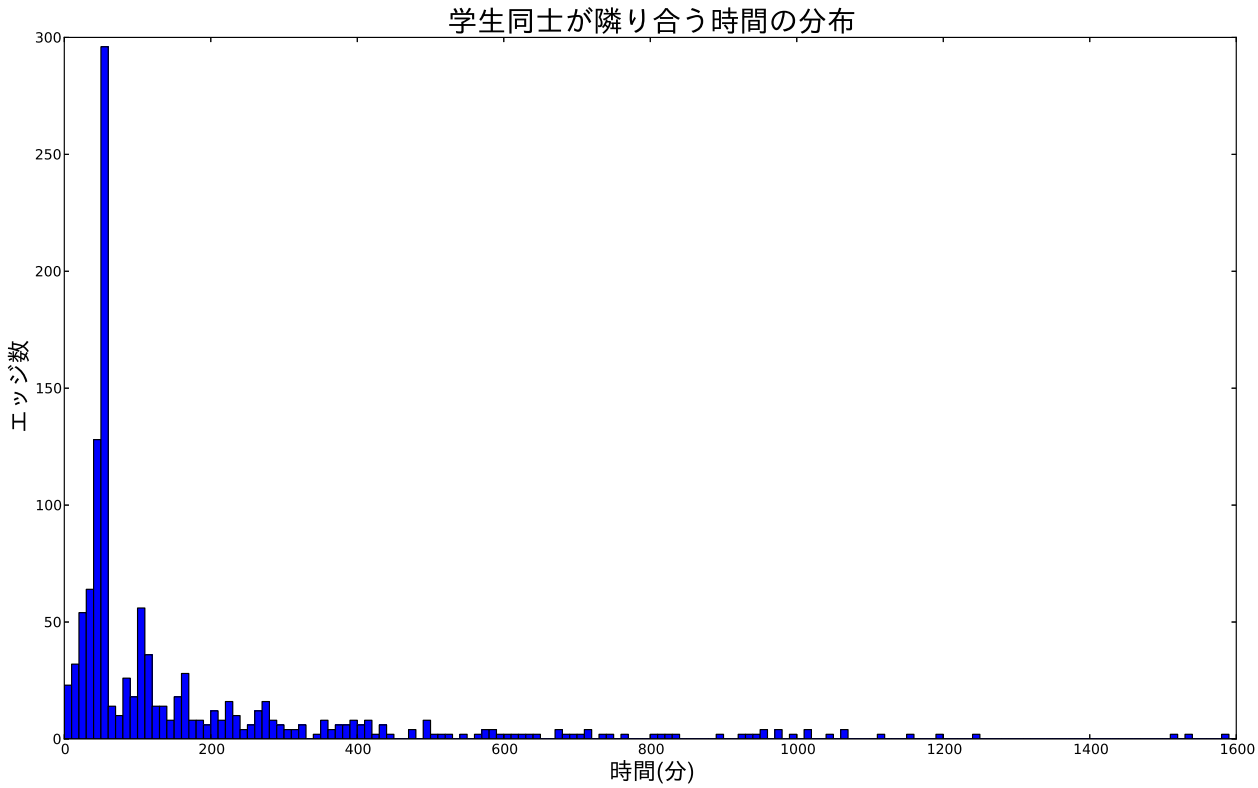

図 3　2013 年 5 月 1 日から 7 月 29 日の 2 年生の情報である。学生同士の隣り合う時間の平均は 164 分である。

4.5 ソーシャルグラフの表示変更

交友関係を把握しやすいように、ソーシャルグラフの表示を変更した。

4.5.1 友達表示

学生 1 人の交友関係を把握したい場合、全ノードが同じ色だと交友関係が把握しにくい。指定学生の交友関係を把握しやすくするために、ノードを色付けし明確にした。指定学生の色を変更したソーシャルグラフを**図 4** に示す。

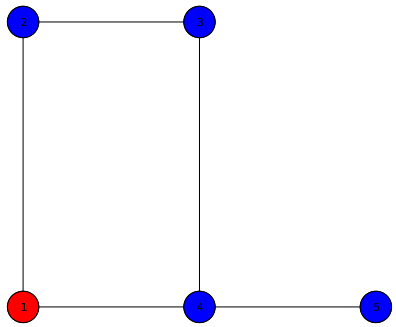

図 4　指定学生をノード 1 とした色の変更画像。赤色の円は指定学生を、青色の円はそれ以外の学生をそれぞれ表わす。円を繋ぐ線は交友関係の存在を表わす。

指定学生とエッジでつながれている学生を友達と認識して、色を変更した。指定学生の友達のノード色変更したソーシャルグラフを **図 5** に示す。

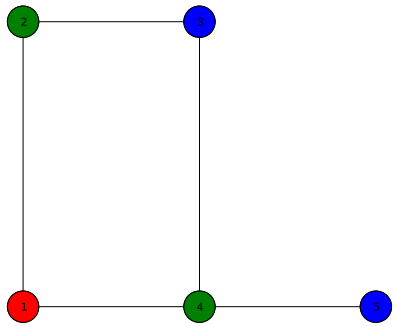

図 5　指定学生をノード 1 とした友達の色変更画像。赤色の円は指定学生を、緑色の円は友達の学生を、青色の円はそれ以外の学生をそれぞれ表わす。円を繋ぐ線は交友関係の存在を表わす。

そして、グラフ上では表示されていないが友達である可能性のある人物を表示させるために以下の方法を取った。対象学生の友達と二人以上つながっている学生を友人である可能性のある人物と判断し色を変更したソーシャルグラフを **図 6** に示す。

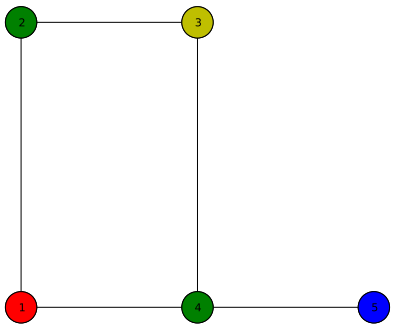

図 6　指定学生をノード 1 とした指定学生の友達の可能性がある学生の色変更画像。赤色の円は指定学生を、緑色の円は友達の学生を、黄色の色は友達の可能性のある学生を、青色の円はそれ以外の学生をそれぞれ表わす。円を繋ぐ線は交友関係の存在を表わす。

4.5.2　**エッジの色の変更**

ノード同士をつなげるエッジがすべて同一では、繋がりが強弱が判断できない。繋がりの強弱を判断できるようにエッジの色を変更した。エッジの持つ時間が全体の平均より 1.5 倍以上で一定の値を持つものを赤色、それ以外を青色で表示する。表示の例を **図 7** に示す。

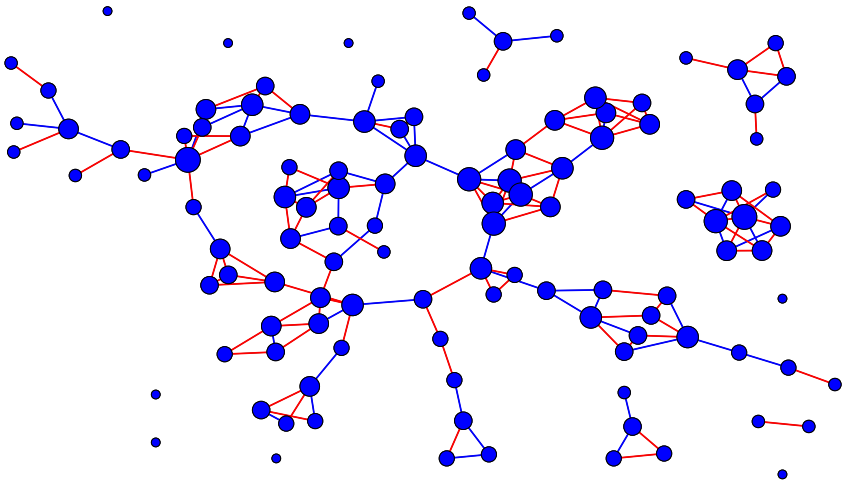

図 7　エッジの色変更画像

4.5.3　**グループの判定**

学生のグループを把握するために、グループを判定し色を変更した。

グループ判定の動作を下記に示す。

1. グラフに表示されている学生 1 人を対象に取る。

2. 対象学生を含む共通の友達が 2 名以上の学生を対象学生が所属しているグループのメンバーと判断する。
3. グループのメンバーと 2 名以上友達の学生をグループのメンバーと判断する。
4. グループに判断されていない学生を対象に取り、再度動作を実行する。

グループ判定を行ったソーシャルグラフを **図 8** に示す。

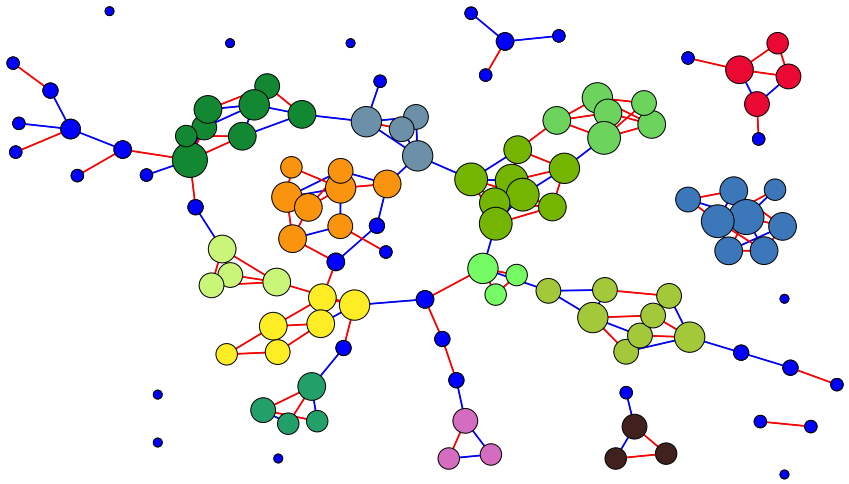

図 8　学生グループの色変更画像

5　考察

出力したソーシャルグラフを研究室メンバーの 1,2 回生 8 人に協力を得て、検証した。その結果から本システムにおける利点と問題点の自説を述べる。

5.1　利点

学生の交友関係を把握することができる。演習室に行われる講義での交友関係に置いて、学生の交友関係の可視化に成功したと言える。

5.2　問題点

交友関係の誤判定が存在する。友達でない学生同士が偶然隣り合い、その時間がエッジの表示条件を超えることが多々存在する。それにより交友関係のない学生同士がエッジでつながれることがある。逆に、様々な学生と交友関係があるために蓄積される時間が均一のためにエッジの表示条件を満たさず表示されないことがある。このため、上記のような誤判定が発生すると考えられる。

6 まとめ

ログイン情報を用いたソーシャルグラフで、学生の交友関係の把握できることは判ったが、改善点がある。
今後の課題として以下の項目が挙げられる。

- ソーシャルグラフの時系列アニメーション
- ソーシャルグラフの精度向上
- ソーシャルグラフの表示変更
- Web アプリケーション化によるリアルタイム表示

謝辞

本研究に関わるきっかけを与えてくださった大垣斉准教授には深く感謝致します。本研究を進めるにあたり御指導及び御協力いただいた東川 諒央講師、情報教育システム研究室のメンバー及び卒業生の方々に深く感謝の意を表します。

参考文献

[1] Brad fitzpatrick. http://bradfitz.com/.

[2] 一輝大向. ソーシャルグラフ. 映像情報メディア学会誌 : 映像情報メディア = The journal of the Institute of Image Information and Television Engineers, Vol. 65, No. 8, pp. 1161–1165, aug 2011.

[3] 稲増文夫, 海部美知. ソーシャルグラフの活用可能性について, july 2009.

[4] Networkx. http://networkx.github.io/.

付録 A　ソースコード

A.1　main.py

```
# -*- coding:utf-8 -*-
import sqlite3
import re
import sys
import datetime
from graph import Graph
from collections import Counter
from collections import defaultdict

class DbRead(object):
 def __init__(self):
  self.con = sqlite3.connect('../db/rwho.db')
  self.from_db = self.con.cursor()
  self.to_db = self.con.cursor()
  self.average_relation = 0
  self.g = GraphCreate()
  self.relation_dict = defaultdict(dict)
  self.average_list = defaultdict(int)
  self.average_time = defaultdict(int)
  self.target_link = defaultdict()
  self.target_list = []
  self.target_area = ''
  self.loaded_target = []

 def read_main(self):
  data = '2013/05/01-2013/07/29'
  self.target = ['s13h___']
  self.select_area(self.target)
  type = self.calculat_type()
  for t in self.target:
   self.target_list.append(t)
  data = re.split('-', data)
  data = self.datetime_calculat(data)
  self.average_calculat(data, type)
  self.write_loop_day(data, type)

 def select_area(self, target):
  area = target[0][0:4]
```

```
 for t in target:
  if not area == t[0:4]:
   self.target_area = 's__%___'
   break
 else:
  self.target_area = area + '___'

def calculat_type(self):
 #total type decision
 type = raw_input('total data or 1 week data(t/w):')
 if type == 't': type = 'total'
 else: type = 'week'
 return type

def datetime_calculat(self, date):
 """ day plus """
 data=[]
 for d in date:
   data.append(datetime.datetime.strptime(d, '%Y/%m/%d'))
 else: data.append((data[1] - data[0]).days+1)
 return data

def set_day(self, day):
 """ data set and return patern """
 return day.strftime('%Y/%m/%d')

def sum_relation(self, name_from, name_to, value):
 """ friend relation sum """
 sum_relation = Counter(self.relation_dict[name_from])
 new_relation = Counter({name_to:value})
 self.relation_dict[name_from] = dict(sum_relation + new_relation)

def write_loop_day(self, data, type):
 """ date loop """
 old_day = data[0]
 relation = 0
 for x in xrange(data[2]):
  day = data[0] + datetime.timedelta(days=x)
  if not (day.weekday() == 2 and self.target_area == 's13h___'):
   for t in self.target_list:
    self.write_edge(t, self.set_day(day))
  self.loaded_target = []
  if x % 7 == 0 and type == 'week' :
```

```
    self.g.show_graph(self.target, type, old_day, day)
    self.g.clean_graph()
    old_day = day
  else:
   self.g.show_graph(self.target, type, old_day, day)

def write_edge(self, target, day):
 """ 1 day loop """
 self.db_load_from(day, target)
 for f in self.from_db:
  if not f[0] in self.loaded_target:
   self.loaded_target.append(f[0])
   self.db_load_side(f)
   self.loop_to(f[0], f[2], f[3], f[4], 1)
 else:
  self.g.remove_edge(self.average_list[day])

def set_new_target(self, target, new_target):
 """ cheak new target and  insert list """
 if target in self.target:
  if not new_target in self.target_list:
   self.target_link[new_target] = target
   self.target_list.append(new_target)

def loop_to(self, from_name, day, from_login, from_logout, relation_value):
 for t in self.to_db:
  relation = self.relation_value(from_login, from_logout, t[1], t[2])
  self.sum_relation(from_name, t[0], relation*relation_value)
  if self.average_list[day] <= self.relation_dict[from_name][t[0]]:
   self.g.add_edge(from_name, t[0], weight=self.relation_dict[from_name][t[0]])
   self.set_new_target(from_name, t[0])

def average_calculat(self, data, type):
 """ cheak and friend relation average calculat """
 def count(valueunt_edge,count_node):
  return all_retation, all_time,count_edge,count_node
 for x in xrange(data[2]):
  day = self.set_day(data[0] + datetime.timedelta(days=x))
  if x == 0 or (x % 7 == 0 and type == 'week'):
   all_relation = 0
   all_time = 0
   count_edge = []
   count_node = []
```

```
  if not (data[0] + datetime.timedelta(days=x) == 2 and self.target_area == 's13h___'):
   self.db_load_from(day)
   for f in self.from_db:
    if f[0] not in count_node: count_node.append(f[0])
    self.db_load_side(f)
    for t in self.to_db:
     all_relation += self.relation_value(f[3], f[4], t[1], t[2])*1
     all_time += (f[4] - f[3]) + (t[2] - t[1])
     if not [f[0],t[0]] in count_edge:
      count_edge.append([f[0],t[0]])
     if not t[0] in count_node:
      count_node.append(t[0])

  if len(count_edge) != 0:
   self.average_list[day] = all_relation/len(count_edge)
  if len(count_node) != 0:
   self.average_time[day] = all_time/len(count_node)

def db_load_from(self, data, target='s__%___'):
 """ target and date wrild card """
 t=(data,target)
 self.from_db.execute(u"select name,rwho.number,data,login,logout,seki.mae,seki.usiro,migi,hidari,seki

def db_load_side(self,from_data):
 """ database load target said seki """
 t=(from_data[2],from_data[3]+1,from_data[4]-1,from_data[3]+1,from_data[4]-1,
    from_data[5],from_data[6],from_data[7],from_data[8], self.target_area)
 self.to_db.execute(u"select name,login,logout from rwho where data=? and (login between ? and ? or lo

def db_load_nextside(self,from_data):
 """ database load one said seki """
 t=(from_data[2],from_data[3]+1,from_data[4]-1,from_data[3]+1,from_data[4]-1,
    from_data[9],from_data[10],self.target_area)
 self.to_db.execute(u"select name,login,logout from rwho where data=? and (login between ? and ? or lo

def db_load_slant(self,from_data):
 """ database load slant or back seki """
 t=(from_data[2],from_data[3]+1,from_data[4]-1,from_data[3]+1,from_data[4]-1,
    from_data[11],from_data[12],from_data[13], from_data[14],self.target_area)
 self.to_db.execute(u"select name,login,logout from rwho where data=? and (login between ? and ? or lo

def relation_value(self,login_from, logout_from, login_to, logout_to):
 """ friend relation clecletion """
```

```
  login = login_from
  logout = logout_from
  if login < login_to: login = login_to
  if logout > logout_to: logout = logout_to
  return logout - login

if __name__ == '__main__':
 d = DbRead()
 d.read_main()
```

A.2 graph.py

```
# -*- coding: utf-8 -*-
import random
import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict

class Graph(object):
 def __init__(self):
  self.g = nx.Graph()
  self.ims = []
  self.count = 1
  self.average_value = 0
  self.ave_value = 1.3

 def add_edge(self,from_node, to_node, weight):
  self.set_node(from_node)
  self.set_node(to_node)
  if not self.g.has_edge(from_node, to_node):
   if not self.g.has_edge(to_node, from_node):
    self.g.add_edge(from_node, to_node, weight=weight)
  else:
   self.g[from_node][to_node]['weight'] += weight - self.g[from_node][to_node]['weight']

 def set_node(self,node):
  if not self.g.has_node(node):
   self.g.add_node(node)

 def clean_graph(self):
  self.g.clear()
```

```
def remove_edge(self,average):
 self.average_value = average
 for t,p in self.g.edges():
  if self.g[t][p]['weight'] < average*self.ave_value: self.g.remove_edge(t, p)

def node_size(self):
 size_list = defaultdict()
 for n in self.g:
  size = 100 + self.g.degree(n)*100
  size_list[n] = size
 return size_list

def draw_color_edge(self,pos):
 target_list = []
 edge_list = []
 for t,p in self.g.edges():
  if self.g[t][p]['weight'] > self.average_value*self.ave_value:
   target_list.append([t,p])
  else:edge_list.append([t,p])
 nx.draw_networkx_edges(self.g, pos, alpha=0.7, edgelist=edge_list, edge_color='b', width=2)
 nx.draw_networkx_edges(self.g, pos, alpha=0.9, edgelist=target_list, edge_color='r',width=2)

def target_node(self, target_list):
 target = []
 for t in target_list:
  if not '_' in t:
   self.set_node(t)
   target.append(t)
 return target

def position_layout(self, target):
 if not self.g.has_node(target): target = None
 pos = nx.pygraphviz_layout(self.g, root=target)
 return pos

def possible_seach(self, target_list):
 possible_list = []
 friend_list = []
 for t in target_list:
  if not '_' in t:
   for f in self.g.neighbors(t):
    if not f in target_list and not f in friend_list:
     friend_list.append(f)
```

```
      if f in possible_list:
       possible_list.remove(f)
     for p in self.g.neighbors(f):
      and_list = list(set(self.g.neighbors(t)) & set(self.g.neighbors(p)))
      if len(and_list) > 1 and not p in target_list and not p in possible_list and not p in friend_list
        possible_list.append(p)
 return possible_list,friend_list

def group_cheak(self,pos,node_size):
 def group(target):
  member_list = [target]
  and_list =[]
  for f in self.g.neighbors(target):
   and_list = list(set(self.g.neighbors(target)) & set(self.g.neighbors(f)))
   if and_list and not f in member_list:
    member_list.append(f)
   for p in self.g.neighbors(f):
    and_list = list(set(self.g.neighbors(target)) & set(self.g.neighbors(p)))
    if len(and_list) > 1 and not p in member_list:
     member_list.append(p)
  for m in member_list:
   for n in self.g.neighbors(m):
    and_list = list(set(member_list) & set(self.g.neighbors(n)))
    if len(and_list) > 1 and not n in member_list:
     member_list.append(n)
  return member_list
 all_nodes = self.g.nodes()
 for n in all_nodes:
  g = group(n)
  if len(g) > 1:
   node_color = '#%06x'%(random.randint(0,16**6-1))
   nx.draw_networkx_nodes(self.g, pos, alpha=0.8, nodelist=g, node_size=[node_size[n]*2 for n in g], n
   for r in g:
    if r in all_nodes: all_nodes.remove(r)

def draw_node(self, pos, target):
 def draw(node_list, alpha, color):
     nx.draw_networkx_nodes(self.g, pos, alpha=alpha, nodelist=node_list, node_size=[node_size[n] for
 friend, possible = self.possible_seach(target)
 another = list(set(self.g.nodes())-set(friend)-set(target)-set(possible))
 node_size = self.node_size()
 nx.draw(self.g, pos, node_size=[node_size[n] for n in self.g], node_color='w', with_labels=True)
 self.group_cheak(pos,node_size)
```

```
 if another: draw(another, 0.3, 'b')
 if possible: draw(possible, 0.5, 'g')
 if friend: draw(friend, 0.5, 'y')
 if target: draw(target, 0.7, 'r')


def show_graph(self, target_list, type, old_day, day):
 self.fig = plt.figure(figsize=(18,12))
 target = self.target_node(target_list)
 pos = self.position_layout(target_list[0])
 self.draw_node(pos, target)
 self.draw_color_edge(pos)
 font = {'fontname'   : 'Helvetica',
         'color'      : 'k',
         'fontweight' : 'bold',
         'fontsize'   : 14}
 plt.title('%s-%s-%s  -  %s-%s-%s'%(old_day.year, old_day.month, old_day.day, day.year, day.month, day
 file_name = 'ave%.2f_%s%d.png'%(self.ave_value, type, self.count)
 plt.savefig('../png/%s'%file_name)
 print 'save %s'%file_name
 self.count += 1
 plt.show()
```