



# INDEX

1. INTRODUCCIÓ………………………………………………………..….pg 02
2. GRAFS ALEATORIS
   1. Binomial Random Graph
      1. Concepte teòric……………………………………........…pg 03
      2. Implementació de la classe……………………..……........pg 03
   2. Geometric Random Graph
      1. Concepte teòric…………………………………...…….....pg 05
      2. Implementació de la classe…………………….….............pg 05
3. METODOLOGIA
   1. Objectius del projecte…………………………………….....…….pg 08
   2. Estructura i material emprat……………………………....……....pg 08
4. EXPERIMENTACIÓ
   1. Estudi de la connectivitat
      1. Hipòtesi inicial…………………………………...….........pg 11
      2. Descripció de funcions………………………….......….....pg 11
      3. Resultats obtinguts i estadístiques……………..………....pg 15

4.2. Estudi de la component connexa gegant

4.2.1. Hipòtesi inicial………………………………………....….pg 17

4.2.2. Descripció de funcions…………………….…...…………pg 17

4.2.3. Resultats obtinguts i estadístiques……………..…..….…..pg 19

1. APARTAT ADICIONAL
   1. Idea inicial…………………………………………………...….....pg 20
   2. Estructura i material emprat………………………………..….......pg 21
   3. Resultats obtinguts i estadístiques…………………….....…..……pg 22
2. CONCLUSIONS I APRENENTATGE………………………..……..........pg 24
3. BIBLIOGRAFIA I WEBGRAFIA……………………………..….……....pg 26
4. ANNEX……………………………………………....…………….…..….pg 27

# INTRODUCCIÓ

En el present document, hem abordat els principals aspectes de la teoria de la transició de fases, mitjançant un estudi experimental de propietats i mesures relatives a les components connexes, quan el graf que tractem es crea amb un model paramètric de grafs aleatoris.

Primerament, hem procedit estudiant les característiques dels grafs aleatoris *Binomial Random Graph* i *Random Geometric Graph* per a poder implementar un codi en *C++*, que de manera eficient implementi aquests models amb els atributs i funcions corresponents.

Seguidament, refrescant els conceptes de component connexa i component connexa gegant, a més de la cerca amb BFS, hem dissenyat mètodes que ens permetin l’estudi de la transició de fase corresponent a les propietats esmentades anteriorment, en els tipus de grafs aleatoris treballats. Tot això mitjançant algorismes implementats en *C++* i *python*.

Cal aclarir que existeixen modificacions respecte a la primera entrega: a la presentació de les dades que es mostren a les gràfiques s’han acotat intervals diferents dels realitzats anteriorment. A més, hem canviat la variable que estudiem a l’eix vertical de les estadístiques de la mida de la component connexa; això és degut al fet que hem considerat que, per tal d’estudiar la transició de fase dels models i propietats proposats en el projecte, és més apropiat representar percentatges que reflecteixen millor les diferències entre les diverses mides de grafs.

Finalment, només queda observar l’estudi fet i els resultats obtinguts; així com les conclusions preses en el nostre projecte, extretes de l’exploració dels temes centrals en el camp dels grafs aleatoris per demostrar o no l’existència de la transició de fase en les propietats proposades.

# GRAFS ALEATORIS

# 2.1. *Binomial Random Graph*

2.1.1. Concepte teòric

El model de graf aleatori G(n, p), *Binomial Random Graph*, es construeix connectant els nodes aleatòriament, de manera que s’inclou una aresta en el graf amb una probabilitat de *p*,independent entre elles. Per entendre-ho millor, a mesura que augmenta p de 0 a 1, serà més probable que s’hi incloguin grafs amb moltes arestes.

2.1.2 Implementació de la classe

Per a la implementació d’aquest model, hem creat una classe en *C++* que representa el graf aleatori en qüestió. Aquesta classe consisteix en un fitxer *.h* que conté els atributs característics i els mètodes necessaris per a l’estudi. L’estructura general va ser inspirada per un exemple senzill d'implementació de grafs trobat a internet.

Els atributs pel graf *binomial random* són:

int numVertex;

int numComponentsConnexes;

int midaCCGegant;

double probabilitat\_aresta;

vector<vector<bool> > matAdjacencies;

L’atribut numVertex representa la quantitat de nodes que componen el graf.

La variable probabilitat\_arestaindica per a quina probabilitat *p* existirà una aresta entre dos vèrtexs *v* i *u*, on *V =* nodes del graf*.* Aquest valor ha d’estar comprès entre 0 i 1.

Els enters numComponentsConnexesi midaCCGegant, descrits més endavant al treball, s’utilitzen per guardar informació de cada mostra necessària pels experiments.

Finalment, l’exploració sobre les arestes del graf es guarda a la matriu de booleans matAdjacencies*.* Vam escollir utilitzar una matriu d’adjacències (en contraposició a una llista) per dues raons: oferia una certa facilitat l’hora d’inicialitzar les arestes aleatòries i presentava millor compatibilitat amb la implementació de les funcions utilitzades en els experiments, els quals són algorismes que vam implementar a l’assignatura d’EDA.

Els mètodes implementats a la classe, que permeten crear i explorar aquest model de graf aleatori són els següents:

Binomial\_random\_graf(int n, double p);

void afegir\_arestes();

void tractament\_components\_connexes(int vertexInici, int &nodeNoVisitat, vector<bool>& nodesVisitats);

int get\_mida\_component\_connexa\_gegant();

int get\_num\_components\_connexes(int vertexInici);

void graf\_gml();

La funció creadora Binomial\_random\_graf(int n, double p), permet la creació d’una instància G de *n* vèrtexs i probabilitat *p* sense arestes, i assigna aquests valors en els atributs numVertexi probabilitat\_aresta.

Posteriorment, per crear les arestes de forma aleatòria, fem una crida a afegir\_arestes(). Aquest mètode recorre tots els elements de la matriu d’adjacències i per a cada posició *[i][j]* assigna valor *true* amb probabilitat *p*.

Les funcions tractament\_components\_connexes(...), get\_mida\_component\_connexa\_gegant(), get\_num\_components\_connexes(...) són utilitzats per aconseguir les dades que mostrem als gràfics i que serveixen per determinar les transicions de fase corresponents. Es troben més detallades als apartats 4.1.2. i 4.2.2. del projecte.

La classe també disposa de la funció graf\_gml(), que emmagatzema el graf en un fitxer gml. Aquest mètode el fem servir per visualitzar el graf amb la llibreria *matplotlib* de *python* per comprovar que les mostres es creen correctament i les funcions calculen exactament les propietats desitjades.

2.2. *Random Geometric Graph*

2.1.1. Concepte teòric

El tipus G(n, r), *Random Geometric Graph*, és un graf geomètric no dirigit amb nodes que existeixen en un espai en dues dimensions. En aquest model, dos vèrtexs *u*, *v* ∈ V estan connectats si, i només si, la distància entre ells és menor o igual que la donada pel paràmetre radi, *r*.

2.1.2 Implementació de la classe

El model *Random Geometric Graph,* també ha estat implementat mitjançant una classe en *C++* (fitxer .h) on s’han establert els paràmetres i mètodes adients.

La classe en qüestió presenta atributs compartits amb l’anterior model, a més de disposar d’altres atributs addicionals característics d’aquest tipus:

int numVertex;

int numComponentsConnexes;

int midaCCGegant;

int dimensions;

float radi;

vector<posicio> posicioVertexs;

vector<vector<bool> > matAdjacencies;

L’atribut dimensions correspon a una *d* en unitats que conforma un pla d’amplada i alçada . Dins aquest pla és on es crea el graf G(n,r).

El paràmetre radi és el que determina l’existència d’una aresta entre dos nodes qualssevol, també expressat en unitats.

El vector posicioVertexs conté totes les posicions dels nodes del graf en els eixos *x* i *y* del taulell, emmagatzemades mitjançant un struct de dos enters.

Els atributs numVertex, numComponentsConnexes, midaCCGegant, matAdjacenciessón atributs generals presents també en la classe de graf aleatori anterior, que fan la mateixa funció.

Els mètodes propis que hem implementat per treballar amb el *Random Geometric Graph* són:

Random\_geometric\_graf(int n, float r, int dim);

void afegir\_arestes();

void tractament\_components\_connexes(int vertexInici, int &nodeNoVisitat, vector<bool>& nodesVisitats);

int get\_mida\_component\_connexa\_gegant();

int get\_num\_components\_connexes(int vertexInici);

void graf\_gml();

La funció creadora Random\_geometric\_graf(int n, float r, int dim) crea una instància del graf G(n, r) amb numVertexs *= n* i radi *= r*. Els vèrtexs són distribuïts de forma aleatoria en un espai de dimensions .

Per afegir les arestes del graf cridem a la funció afegir\_arestes(). Aquest mètode calcula el mòdul de la distància entre cada parell de vèrtexs pertanyents al graf i el compara amb l’atribut *radi*. Si aquesta distància és més petita l’atribut, llavors direm que existeix una aresta entre ambdós nodes, és a dir, matAdjacencies[i][j] *= true*, altrament, matAdjacencies[i][j] *= false*.

Les funcions tractament\_components\_connexes (...), get\_mida\_component\_connexa\_gegant(), get\_num\_components\_connexes(...) també són compartides amb el model anterior, i estan explicades als apartats 4.1.2. i 4.2.2.. Per últim, aquesta classe també disposa del mètode graf\_gml(), que recull les dades del graf en format gml.

# METODOLOGIA

# 3.1. *Objectius del projecte*

Els objectius d’experimentació d’aquest projecte que ens hem marcat des d’un principi són: conèixer què és una transició de fase i saber reconèixer-la en el moment de l’observació dels resultats, és a dir, trobar el punt on existeix un canvi pronunciat en creixement de la funció que depèn del paràmetre característic de cada model de graf (*p* pel *Binomial Random Graph* i *r* pel *Random Geometric Graph*).

Per a poder observar aquestes transicions de fase hem implementat els materials pertinents i hem fet els experiments amb grafs de diferent grandària fins a tenir una certa evidència de l’existència o inexistència d’una transició de fase per a diferents propietats dels models.

Finalment, referent al hardware emprat per a la compilació i l’execució dels programes usats, hem fet servir un ordinador portàtil amb un processador Intel® Core™ i5-3210M CPU @ 2.50GHz × 4 i amb sistema operatiu *ubuntu 16.04*.

# 3.2. *Estructura i material emprat*

Per tal de realitzar els experiments proposats al projecte, hem dissenyat una estructura concreta amb la qual obtenim les mostres dels grafs aleatoris i les processem. Aquesta estructura permet obtenir els resultats en forma de gràfics de manera automàtica quan executem el nostre programa *main*. Per aconseguir-ho hem utilitzat els llenguatges *C++* i *python* de forma conjunta. D’aquesta manera, dividim l’execució en dues parts; la primera part correspon al treball que fa el codi en *C++* i la segona a aquell realitzat mitjançant *python*.

Primera part: *Càlcul i emmagatzematge de les dades en un fitxer de text.*

El codi encarregat d’aquesta part de l’experimentació es troba al programa *main.cc*. La informació que calcula són parells de valors *X* i *Y* que seran utilitzats als gràfics, on *X* és un valor de *p* o *r*, depenent del model de graf, i *Y* és el valor de la propietat que volem estudiar per a cada *p* o *r* donats.

Cada mostra (*X*,*Y*) es calcula prenent la mitjana de la propietat *Y* per a certa *X* constant a 20 mostres de grafs (paràmetres de creació iguals). Cal puntualitzar que, a mesura que experimentàvem, vam decidir acotar el domini de la component X dels gràfics (eix horitzontal) per probabilitats entre [0 - 0.5], i per radis d’entre [0 - 30], tenint en compte que els grafs estaven disposats en un taulell de dimensió 50 x 50 unitats en el cas dels grafs *random geometric*; més enllà d’aquests intervals els valors d’*Y* (eix vertical) s’estabilitzaven i eren constants.

El valor de *p* s’incrementa 0’02 unitats en cada mostra, per tant, calculem 26 mostres. El valor de *r*, en canvi, s’incrementa en 1 unitat fins a arribar a 31 mostres. Finalment, aquest procés es repeteix 5 vegades per cada graf amb vèrtexs *n =* {20, 40, 60, 80, 100}.

En resum, els experiments requereixen la creació de:

(*Binomial Random Graph*)

(*Random Geometric Graph*)

A mesura que es van obtenint les mostres (*X,Y*), aquestes es van emagatzemant en un fitxer de text. Aquest fitxer té el següent format:

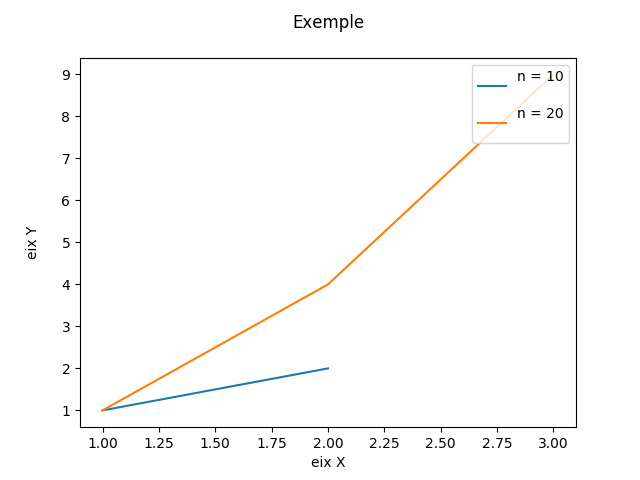
Cada fitxer correspon a un graf individual. Es comença assignant valors generals dels gràfics:

|  |
| --- |
| Nom\_grafic.txt |
| Títol del gràfic  Etiqueta de l’eix X  Etiqueta de l’eix Y  nombre de n diferents  (salt de línia adicional) |

Seguidament, es recullen les mostres de cada conjunt de grafs amb nombre de vèrtexs iguals. El bloc de text a continuació es repeteix “*nombre de n diferents”* vegades:

|  |
| --- |
| nombre de vertexs del graf  nombre de mostres X,Y  X1 Y1  X2 Y2  …  (salt de linia adicional) |

A continuació, presentem un exemple de fitxer i el gràfic que genera:



|  |
| --- |
| exemple\_grafic.txt |
| Exemple  eix X  eix Y  2  10  2  1 1  2 1  20  3  1 1  2 4  3 9 |

Segona part: *Lectura del fitxer i creació dels gràfics corresponents.*

Un cop s’han creat els fitxers, s’executa l’*script* de *python* *programaGrafics.py*. Aquest programa s’encarrega de llegir els fitxers, crear els gràfics corresponents i guardar-los com imatges.png a la carpeta “grafics”.

# EXPERIMENTACIÓ

4.1. *Estudi de la connectivitat*

4.1.1. Hipòtesi inicial

Abans de realitzar l’experiment esperàvem aconseguir unes gràfiques que s’equiparessin a la gràfica donada com a exemple a l’enunciat del treball, on les línies que mostren la probabilitat de ser connexos tenen un recorregut molt suau i progressiu. A més, al veure la imatge, confiàvem que els valors entre els quals es trobaria la transició de fase del graf de tipus *binomial random* fossin similars, és a dir, entre probabilitats de 0.1 i 0.3. Per l’altre model, *Random Geometric Graph,* esperàvem un comportament semblant al primer tipus de graf.

4.1.2. Descripció de funcions

Per tal d’estudiar la k-connectivitat dels models de grafs aleatoris, hem implementat funcions i atributs dins de cada classe que ens permetin tractar els elements necessaris per calcular el nombre de components connexes de cadascun:

**int numComponentsConnexes**

Atribut enter contingut a cada graf que representa el nombre de components connexes d’aquest. S’inicialitza a 1 i, amb les funcions explicades a continuació, es va incrementant a mesura que trobem una nova component connexa dins el graf estudiat.

**void tractament\_components\_connexes(int vertexInici, int &nodeNoVisitat, vector<bool> &nodesVisitats)**

Funció que rep com a paràmetres un vèrtex inicial des d’on començar la cerca, una variable que indica un node no visitat, i un vector de booleans que representa les visites als vèrtexs. Amb aquests elements, el mètode realitza una cerca aplicant BFS (Breadth-First Search) i recorre tots els vèrtexs que formen part de la mateix component connexa, marcant-los com a “visitats”. Un cop fet el recorregut, la funció fa el tractament del vector de nodes visitats i busca si n’hi ha algun que no formava part de la component connexa; en cas afirmatiu, es guarda aquest vèrtex a la variable nodeNoVisitat i finalitza la seva cerca.

La fita superior del cost temporal teòric d’aquesta funció ve donat pel cost de l’algorsisme BFS, que és , i del recorregut posterior del vector nodesVisitats amb mida n, que és en el pitjor cas ja que se’l recorreria sencer. Per tant, el cost total de la funció és la suma d’aquests dos costos, + , que equival a .

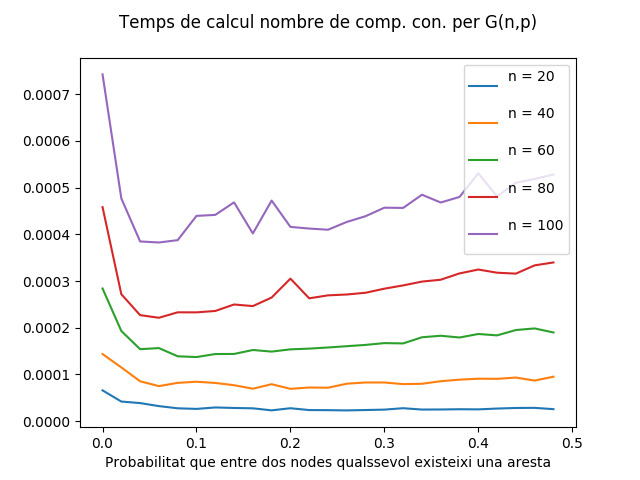
**int get\_num\_components\_connexes(int vertexInici)**

Funció que, donat un vèrtex inicial qualsevol, calcula i retorna un enter que indica el nombre de components connexes del graf corresponent. Dins aquest mètode, es fa una crida a la funció tractament\_components\_connexes(...) proporcionant-li com a paràmetres el vèrtex inicial, un enter que indica un vèrtex no visitat fins al moment (inicialitzat a -1), i una estructura amb els nodes visitats. A continuació, la funció realitza les operacions pertinents, esmentades anteriorment . Un cop finalitzada, el primer mètode disposa de la variable nodeNoVisitat per comprovar si encara existeix un node no visitat i, en aquest cas, tornar a fer la crida a tractament\_components\_connexes(...), aquest cop amb el node no visitat com a node inicial. D’aquesta manera, cada vegada que ha de tornar a recórrer a la segona funció s’incrementa el nombre de components connexes del graf (numComponentsConnexes).

Considerem que el cost temporal d’aquesta funció és , sent *k* una constant que representa el nombre de nodes no visitats diferents (≈ nombre de components connexes diferents). La variable *ni* representa el nombre de vèrtexs de la component connexa del graf G i *mi* el nombre d’arestes d’aquest. Per tant, el cost acaba sent el del conjunt dels BFS realitzats, .

Tots aquests elements i mètodes es troben tant a la classe Binomial\_random\_graf com a la classe Geometric\_random\_graf i es poden veure més detalladament en els codis presents a la carpeta.

Per a verificar les hipòtesis sobre els costos temporals esperats d’aquestes funcions, hem monitoritzat el temps d’execució del mètode get\_num\_components\_connexes(...) dins el programa *main.cc*. Els resultats obtinguts després de fer córrer el programa els hem processat utilitzant gràfics, perquè es vegi el comportament d’una manera més visual:

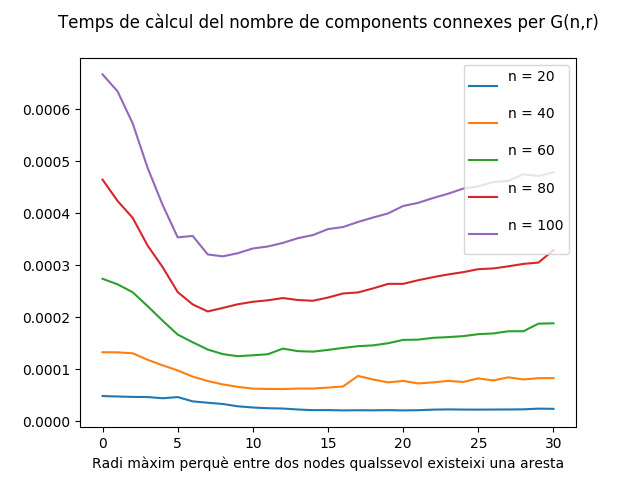
***Binomial Random Graph***

En aquesta gràfica podem observar que els grafs amb valors petits de *p*, presenten un cost temporal elevat, però de seguida cau en picat fins a estabilitzar-se aproximadament a *p =* 0.03-0.04. A partir d’aquest punt, tot i que existeixen certes irregularitats, que creiem que són degudes al fet que el processador ha de dedicar cert temps a altres tasques no relacionades amb l’experiment, el creixement del cost sembla adquirir un comportament lineal, proporcional al valor de *p*.

Creiem que el primer fenomen és degut a que, al haver-hi un nombre petit d’arestes, existeixen moltes components connexes, per tant, considerem i on . Això significa que hi haurà un nombre de crides a la funció tractament\_components\_connexes(...) propera a *n* que, encara que el mètode en aquest cas tingi cost , el fet de realitzar les pròpies crides pot ser molt costós.

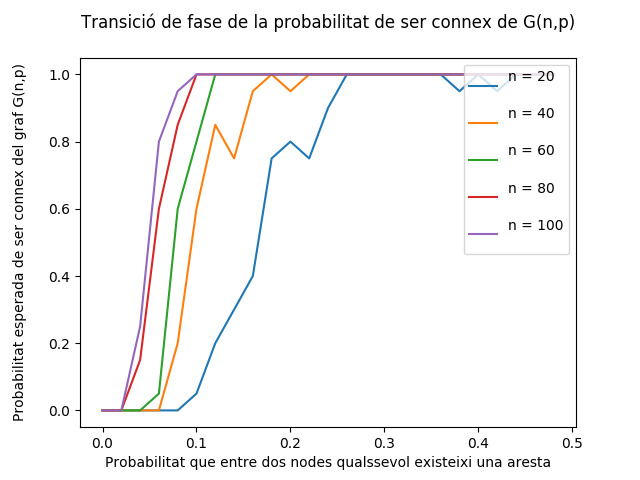
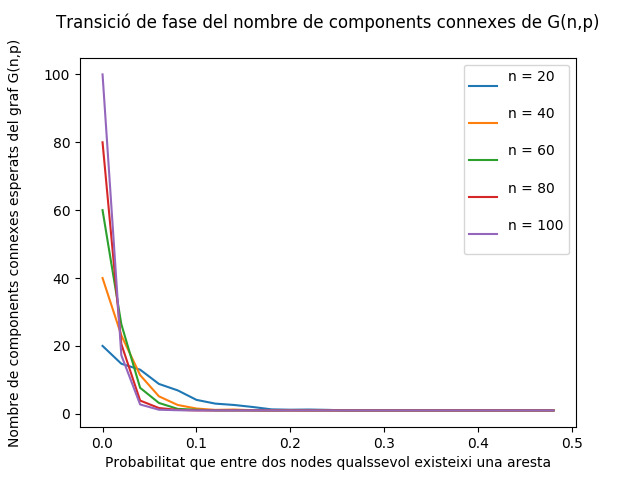
La baixada pronunciada del cost podria atribuir-se a què el nombre de components connexes també baixen en picat a partir d’un cert punt i, conseqüentment, també ho fan el nombre de crides a tractament\_components\_connexes(...).

Finalment, trobem l’augment lineal del cost temporal de la funció. Pensem que pot ser degut al fet que, un cop el nombre de components connexes es proper a 1, tan sols existeix una única crida a la funció tractament\_components\_connexes(...), per tant, el pes del cost recau en el BFS que realitza aquest mètode. Per cada graf, la *n* és constant i només varia el valor de la *m* que va augmentant linealment a causa de l’increment de *p*, .

***Geometric Random Graph***

Sembla presentar les mateixes característiques que les que es donen al model anterior. Trobem la baixada pronunciada al principi, començant amb costos més alts i, seguidament, observem un creixement lineal que varia en funció del valor *r.*

4.1.3. Resultats obtinguts i estadístiques

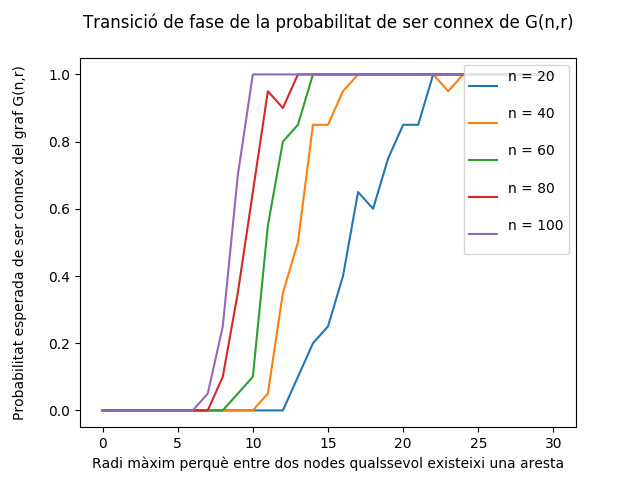
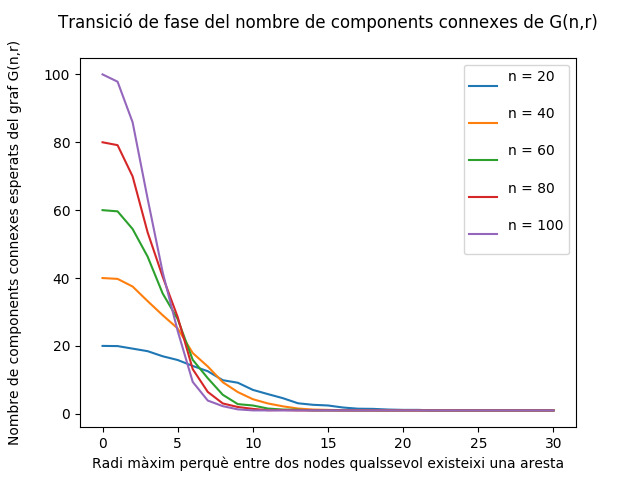
*BINOMIAL RANDOM GRAPH*

Pel *Binomial Random Graph,* podem observar que la gràfica del nombre de components connexes no sembla contenir un punt de transició clar; en els grafs de 20 nodes podem arribar a diferenciar una baixada lineal entre els valors de *p* d’entre 0 i 0.15 abans d’arribar al punt de *p* on el graf sempre és connex, però per valors superiors, la funció decreix més ràpidament, ja que el nombre de components connexes de seguida cau en picat fins que el graf és totalment connex.

La imatge de la probabilitat de ser connex, tot i mostrar la mateixa propietat que la gràfica proporcionada per l’enunciat del treball, no traça la mateixa progressió en les línies. L'evolució de la connectivitat sembla ser més abrupta, com menys nodes té el graf més irregular és. També podem observar que els punts de transició comencen més aviat que en la imatge de l’enunciat, que són compresos entre els valors de *p* de 0.02 i 0.11, en contrast a l'interval [0.1 - 0.3], respectivament.

No obstant, tot i que els intervals de compressió de la transició de fase no siguin els mateixos, la constant de creixement o decreixement per diferents mides de grafs es compleix. És a dir, per grafs amb menor nombre de nodes, necessiten probabilitats més altes per tenir un sol component connex, i per tant quan aquesta grandària s’incrementa, la connectivitat dels vèrtexs és més fàcil d’aconseguir amb valors més petits de *p.* Així com per a tots els valors de n trobem un punt on la seva evolució sempre serà un graf connex.

*RANDOM GEOMETRIC GRAPH*



Abans de comentar les gràfiques, cal recordar que totes les mostres d’aquest model s’han creat en un taulell de 50x50 unitats, independentment de la mida del graf.

Les estadístiques del *Random Geometric Graph,* sí que semblen presentar un punt de transició de fase, a diferència del gràfic de components connexes del model anterior. Aquest sembla començar a un valor aproximat de *r* de 2 unitats per totes les mides de grafs; abans d'aquest moment el nombre de components connexes baixa de forma subtil, i al creuar aquest valor, la baixada es torna més pronunciada, sobretot quan augmenta la grandària del graf. A partir d’aquest punt els nodes comencen a connectar-se entre ells, disminuint el nombre de components connexes de forma considerable a cada unitat que creix el radi. Finalment, la caiguda es frena de forma progressiva i s’arriba a la connectivitat total. Cal destacar que aquest interval és diferent per cada mida de graf: com més nodes té, més aviat i ràpida és la frenada. Així, notem que mentre el graf amb 20 vèrtexs experimenta una caiguda més aviat lineal i arriba al punt màxim de connectivitat a unes 14 unitats, el graf amb 100 nodes ja és connex quan el radi necessari sobrepassa les 6-7 unitats.

La gràfica de la probabilitat de ser connex té el mateix caire irregular que el del model esmentat anteriorment. Les discontinuïtats es veuen amb més claredat als grafs amb menor nombre de nodes i es suavitzen amb valors més grans. A aquesta imatge també és evident la transició de fase envers la connectivitat dels diferents grafs. El punt exacte coincideix al moment en el qual la primera gràfica arriba a la connectivitat absoluta del graf: la mostra de mida 20 canvia radicalment al punt 13-14, mentre que el graf de 100 nodes té el seu punt de transició als valors 6-7.

4.2. *Estudi de la component connexa gegant*

4.2.1. Hipòtesi inicial

Per a les gràfiques sobre la mida de la component connexa gegant esperàvem trobar una transició de fase semblant a la de les altres propietats, tant pels grafs *binomial random* com pels *random geometric*, on existeix una regió pre-transició en la qual el valor no canvia de forma considerable i una regió post-transició on la mida de la component gegant és igual a la mida del graf, és a dir, és màxima, i no pot fer-se més gran.

4.2.2. Descripció de funcions

Per tal d’estudiar la component connexa gegant de cada graf i el seu comportament, hem implementat funcions i atributs dins cada classe que ens permeten observar la mida de cada component connexa d’un graf i seleccionar-ne la més gran:

**int midaCCGegant**

Atribut que representa la mida de la component connexa gegant del graf corresponent. Aquest enter es va modificant mitjançant els mètodes següents en relació al nombre de nodes que conté la component connexa més gran trobada fins al moment.

**void tractament\_components\_connexes(int vertexInici, int &nodeNoVisitat, vector<bool>& nodesVisitats)**

Funció esmentada en l’apartat 4.1.2. utilitzada pel tractament de les components connexes d’un graf. Rep com a paràmetres: el vèrtex inicial des d’on es vol començar la cerca BFS, una variable que representa un node no visitat del graf, i el vector de vèrtexs visitats; amb aquests paràmetres realitza les intervencions per estudiar cada component connexa, analitzar la mida d’aquestes i mitjançant una comparació emmagatzemar la grandària màxima a l’atribut midaCCGegant.

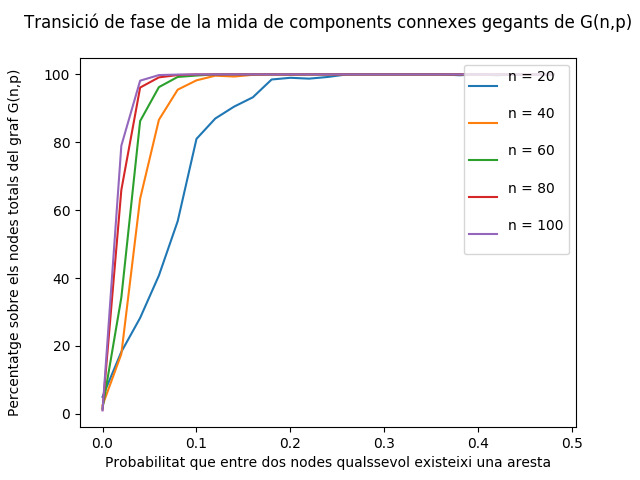
**int get\_mida\_component\_connexa\_gegant()**

Funció que realitza un retorn de l’atribut midaCCGegant un cop han sigut tractades les components connexes del graf. És important que, abans de fer la crida a aquest mètode, s’hagi executat la funció anterior tractament\_components\_connexes(...), que està continguda al *getter* get\_num\_components\_connexes(). Per tant, si es vol conèixer la mida de la component connexa gegant del graf, és important respectar aquest ordre.

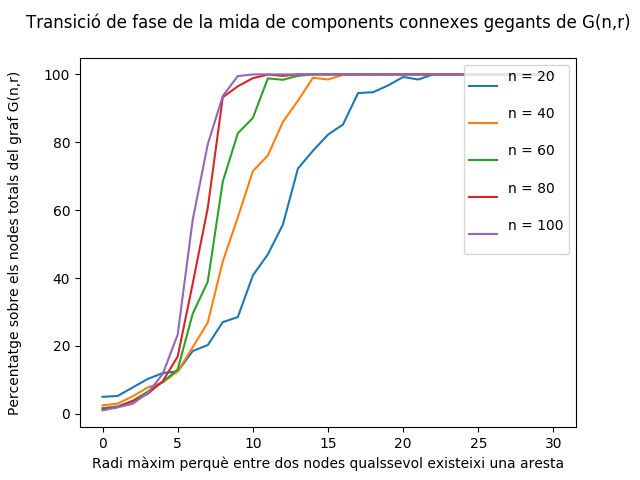
El cost d’aquesta funció és constant, tan sols realitza un retorn d’una variable, ja que la mida del component gegant es calcula quan es crida get\_num\_components\_connexes(...) que en parlem a l’experiment anterior. Per tant el cost és Ө(1).

Tant l’atribut i els mètodes citats, com el funcionament de cadascun es poden veure amb més detall als codis proporcionats dins la carpeta.

4.2.3. Resultats obtinguts i estadístiques



La gràfica del *Binomial Random Graph* per la mida de la component connexa gegant no sembla presentar una transició de fase concreta. Es comporta de la mateixa manera que l’estadística de la connectivitat, però de forma inversa, ja que quan el graf és totalment connex, la mida de la component connexa és màxima. Només podem deduir la propietat que com més petit és el graf, més gran ha de ser el valor de *p* perquè la component connexa gegant comprengui tots els nodes del graf.



A la gràfica del *Random Geometric Graph* sí que es pot observar una transició, aproximadament a *r* = 4*.* Els pendents de creixement s’accentuen amb valors més elevats de *n,* com podem observar a les mostres de 100, 80 i 60 nodes. En aquests grafs, l’assoliment de la mida màxima a la component connexa gegant es troba entre radis de 6-8 unitats. Per altra banda, les mostres amb *n* = 40i *n* = 20 manifesten aquest fet més tard, amb radis de 11 i 15 respectivament; és per aquests valors quan la component connexa gegant està formada per tots els nodes que componen el graf.

# APARTAT ADICIONAL

## 5.1. *Idea inicial*

Aquest apartat podia tractar de l’estudi de la transició de fase d’altres propietats en els mateixos grafs aleatoris o bé de les mateixes propietats però aplicat a algun tipus de graf aleatori diferent.

Nosaltres, hem obtat per aprofundir en l’estudi de la connectivitat i, per tant, hem decidit centrar-nos en l’exploració de la transició de fase del nombre de components connexes, així com la de la probabilitat de ser connex i la mida de la component connexa gegant, aplicat a un altre tipus de graf aleatori.

Després de cercar informació sobre diferents models paramètrics de graf aleatoris, hem triat el model Erdős-Rényi, denotat G(n, m). A diferència dels grafs vistos fins el moment, on el nombre d’arestes era un valor no-determinístic, aquest graf conté un nombre d’arestes que sí està prefixat, i ve determinat pel paràmetre *m*. L'element aleatori d’aquest tipus de graf és la seva topologia, és a dir, la disposició de les arestes entre els diferents nodes.

Resumint, el funcionament del darrer model, el podem veure com un procés de creació, que comença amb un graf de *n* vèrtex i sense arestes, i a cada pas n’afegeix una aresta escollida uniformement de les que falten.

Com a hipòtesi inicial, vam pensar que els gràfics resultants de totes les propietats podien ser similars als obtinguts amb els models aleatoris estudiats fins al moment. Per una banda, podia ser que les transicions de fase fossin poc clares, gairebé inexistents com el cas d’algunes del *Binomial Random Graph* o, per altra banda, podia donar-se el cas que s’assimilessin a les estadístiques obtingudes amb el *Random Geometric Graph*, amb unes transicions de fase més visibles i pronunciades.

## 5.2. *Estructura i material emprat*

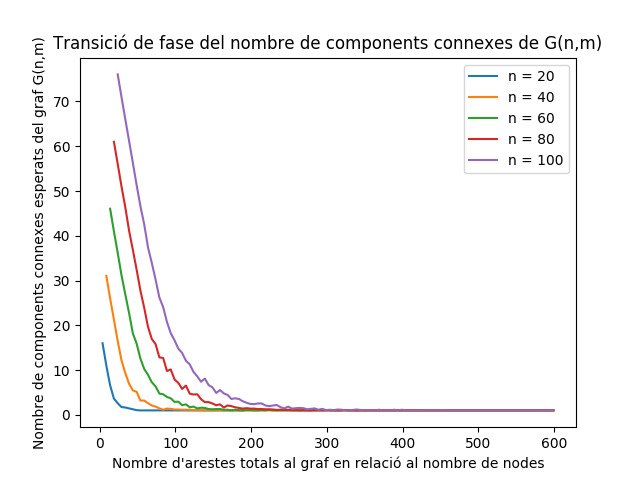
Com que per aquest experiment addicional es podia fer ús de la llibreria *NetworkX* de *python*, hem implementat un sol programa en python perquè generi els grafs, calculi les dades que volem observar i crei els gràfics estadístics corresponents. D’aquesta manera, ens hem estalviat molta complexitat i temps a l’hora de la preparació del material per l’experimentació.

El mètode que usem per recollir dades es similar al de les proves anteriors. L’interval de nombre de nodes del graf que utilitzem és *n* = {20, 40, 60, 80, 100}. Per cada una d’aquestes *n* efectuem mostres amb valors de *m* dins l’interval [] si < 600, o [] altrament. A cada mostra augmentem *m* en 5 unitats.

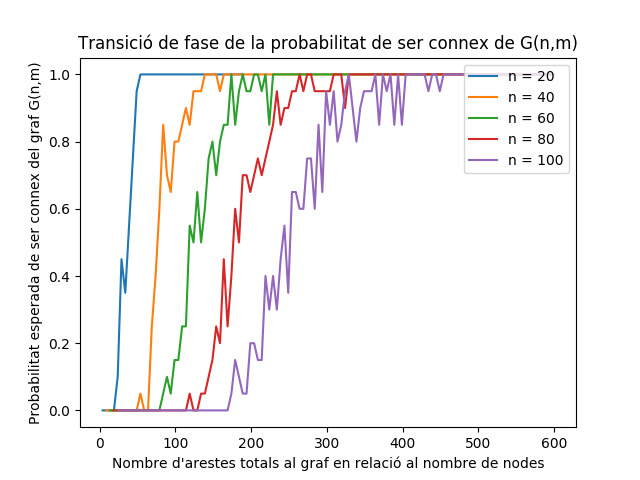
Finalment, per aconseguir cada mostra es creen 20 grafs diferents i es fa la mitjana de la propietat obtinguda en cada un d’ells. El nombre total de grafs creats són:

Les dades obtingudes es van guardant contínuament en dues columnes, la primera conté el nombre d’arestes de la mostra i la segona, el valor de la propietat calculada. Un cop omplert el document, aquests es passa al gràfic corresponent i es guarda en format d’imatge.

## 5.3. *Resultats obtinguts i estadístiques*

**Nombre de components connexes** 

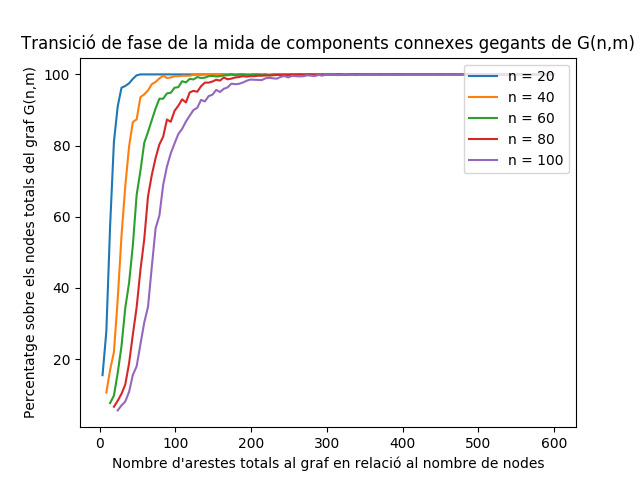
Al gràfic que mostra el nombre de components connexes s'observa cap punt de transició, a mesura que augmenten el nombre d’arestes del graf el nombre de components connexes decreix en picat. La caiguda frena fins que esdevenen grafs completament connexos (el nombre de components connexes és 1). Aquesta frenada sembla ser proporcional a la quantitat de nodes que conformen el graf: com menys nodes hi ha (menor grandària del graf), abans es comença a frenar el decreixement de la funció. Deduïm que les petites oscil·lacions que apareixen poden ser degudes a un fenomen que s’explica pel següent gràfic.

**Probabilitat de ser connex**

Pel que fa al gràfic que estudia la probabilitat de ser connex, sí que sembla contenir una transició de fase bastant més evident. Es pot observar el punt de transició coincideix quan Això pot ser donat que el nombre mínim d’arestes perquè un graf sigui connex ha de ser com a mínim; d’aquesta manera tots els nodes del graf estan connectats en una única component connexa. Per tant, a partir d’aquest valor, la funció creix exponencialment, ja que comencen a aparèixer grafs connexes.

No obstant, per a valors de *n* més grans, sembla que també es pot observar que el camí cap a una probabilitat de 1.0 és molt més irregular que en els experiments realitzats anteriorment. Una possible explicació per aquest fenomen seria l'aleatorietat de la disposició de les arestes en el model de graf G(n,m); és a dir, entre dos grafs amb la mateixa mida *n* i nombre d’arestes similar, tant pot ser que esdevingui connex com no. Tot i així, quan ens fixem en tot el conjunt de valors d’*m*, podem veure una clara tendència cap a la connectivitat absoluta.

**Mida de la component connexa gegant**

Pel que fa a l’estadística de la mida de la component gegant, tampoc sembla presentar cap transició de fase evident.

No obstant és interessant remarcar la característica que hem observat en els gràfics anteriors: la component gegant comprèn el 100% de nodes a partir de valors de *n* molt propers a , per la propietat esmentada anteriorment.

# CONCLUSIONS, VALORACIONS I APRENENTATGE

**Conclusions**

Seguit de l’exhaustiva experimentació, ens hem dedicat a extreure conclusions, entre la part empírica i la teòrica, per tal de sintetizar tot el que hem treballat.

En primer lloc, ens centrarem en l’experimentació de les propietats del model de graf aleatori *Binomial Random Graph*. A l’observar les gràfiques obtingudes dels experiments hem conclòs que tenim evidències per a confirmar que es produeix una transició de fase quan estudiem la propietat de connectivitat; tot i això, cal destacar que aquesta evidència només es veu clara en l’estadística de la probabilitat de ser connex, i no en la del nombre de components connexes del graf. Un altre punt interessant és que per a grafs amb pocs nodes es necessiten probabilitats més altes per aconseguir la connectivitat total; en canvi, quan la quantitat de nodes és més gran, la probabilitat necessària perquè el graf sigui connex disminueix.

Respecte a l’estudi de la component connexa gegant, les gràfiques que ens han proporcionat els experiments, tal com hem comentat al seu apartat, no semblen contenir cap punt de transició. Creiem que pot ser degut al fet que la mida de la component connexa gegant canvia considerablement per qualsevol valor de *p*, i no es manté una mida constant durant cap interval.

Seguidament, per al model de graf aleatori *Random Geometric Graph,* hem obtingut gràfics similars als de tipus G(n, p) per algunes propietats, però diferents per unes altres. A la gràfica de la transició de fase del nombre de components connexes observem un gràfic amb una forma de transició més regular i clara. En complement, la imatge que mostra la probabilitat de ser connex segons el valor *p* també presenta un punt de transició en un moment de la gràfica. Per tot això, concloem que també existeixen evidències per a demostrar que hi ha una transició de fase, en aquest tipus de graf i estudiant aquesta propietat.

En l’estudi de la transició de fase en component connexes gegants per a aquest tipus de model, teníem com a hipòtesi poder observar uns gràfics semblants a les representacions dels experiments de connectivitat anteriors. Un cop realitzades les proves, podem concloure que en aquest model si que existeix una transició de fase per a aquesta propietat, a diferència del graf *binomial random*, i per tant, corroborem la nostra hipòtesi.

Finalment, en el model de graf aleatori addicional, G(n, m), com que sabem que aquest model segueix una distribució semblant al G(n, p), no ens ha sorprès trobar que el gràfic que mostra la transició de fase del nombre de components connexes no presentés cap tipus de transició. En canvi, en el gràfic que reflecteix la probabilitat de ser connex, tot i presentar molt soroll i molta irregularitat, es pot observar una evolució molt semblant a la transició de fase i, per tant, hem conclòs que en aquest tipus de graf també s’hi dóna la transició per la connectivitat.

Per acabar, en l’experiment de la component connexa gegant, hem vist que no es reflecteix cap tipus de transició de fase, similar al cas del *Binomial Random Graph* per aquesta mateixa propietat, i en conclusió, no podem afirmar l’existència d’aquesta.

**Aprenentatge extra**

Gràcies a la documentació trobada d’estudis de diferents propietats de la connectivitat d’aquests tipus de graf, hem après que en el cas del graf tipus G(n, p), tot i que les arestes es trien de forma independent amb una certa probabilitat, hi ha certes propietats globals del graf que sempre es compleixen:

* tenim una constant d que serà p\*n.
* si d < 1, llavors el graf no tindrà components connexes de mida més gran que log(n) vèrtex.
* si d = 1 llavors la component connexa gegant serà de mida n²/³.
* si d > 1, llavors el graf tindrà una única component connexa gegant.
* si la p < (1-e) ln n / n llavors el graf contindrà segur un vertex aïllat.
* si la p > (1+e) ln n / n serà connex.

# BIBLIOGRAFIA I WEBGRAFIA

**Referències per a l’implementació**

[1] Networkx.github.io. *Announcement: NetworkX 2.5 — NetworkX 2.5rc1.dev20191019163954 documentation*. [online] Available at: <https://networkx.github.io/documentation/latest/release/release_dev.html>

[2] Cplusplus.com. Input/output with files - C++ Tutorials. [online] Available at: <http://www.cplusplus.com/doc/tutorial/files/>

[3] Cplusplus.com. *<ctime> (time.h) - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/ctime/>

[4] Matplotlib.org. *Pyplot tutorial — Matplotlib 3.1.1 documentation*. [online] Available at: <https://matplotlib.org/3.1.1/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py>

**Referències a la documentació d’informació**

[1] R. Sedgewick and K. Wayne. Algorithms. Addison-Wesley, 4th edition, 2011.

<https://algs4.cs.princeton.edu/>

[2] M.E.J. Newman. Networks. An Introduction. Oxford University Press, 2010.

[3] Tesliuc, M. RANDOM GRAPHS AND THEIR APPLICATIONS. [pdf] Math.uchicago.edu. Available at: <http://math.uchicago.edu/~may/REU2017/REUPapers/Tesliuc.pdf>

[4] En.wikipedia.org. Erdős–Rényi model. [online] Available at: <https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model>

[5] En.wikipedia.org. Random graph. [online] Available at: <https://en.wikipedia.org/wiki/Random_graph>

[6] Kun, J. *Erdos-Renyi – Math ∩ Programming*. [online] Math ∩ Programming. Available at: <https://jeremykun.com/tag/erdos-renyi/>

[7] Cs.upc.edu. *Random Geometric Graphs*. [pdf] Available at: <https://www.cs.upc.edu/~diaz/TRANSPES/RGG_HK.pdf>

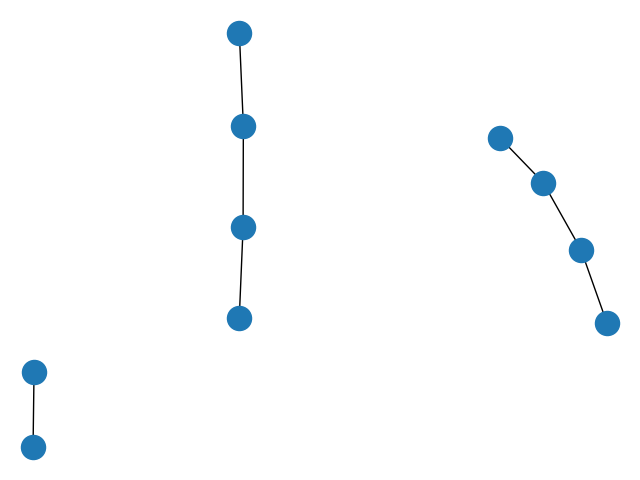
# ANNEX

Com a annex del treball, hem volgut afegir dos codis que ens han sigut útils a l’hora de conformar el projecte. A la carpeta “annex” estan continguts els següents fitxers:

* Un arxiu *main\_manual.cc* que hem fet servir per introduir paràmetres aleatoris de forma manual i generar els grafs G(n,p) i G(n,r) corresponents.
* Un *script* de *python visualitzador.py,* que utilitza les llibreries *NetworkX* i *matplotlib,* implementat per tal de mostrar de forma visual els grafs aleatoris que generem, mitjançant la crida de la funció *graf\_gml()* (pròpia de cada una de les classes),esmentada a l’apartat 2 del treball, i que tradueix la matriu d’adjacències corresponent a notació gml.

Aquests codis ens han guiat a l’hora de comprovar el funcionament i la correctesa de les classes i els mètodes que hem implementat per realitzar els experiments del projecte, és per això que considerem rellevant afegir-los a l’entrega.

Aquí es mostra un exemple d’un dibuix d’un graf qualsevol generat mitjançant el mètode citat:



Exemple de *Binomial Random Graph* G(10,0.1)