Transició de fase en propietats de grafs

Grau A Q2 - Curs 2021-2022

Pablo Vega Gallego

Adrian Cristian Crisan

Mark Smithson Rivas

Índex

1. Objectius

L'objectiu del projecte és analitzar experimentalment l'existència o no de transició de fase en processos de percolació en grafs. En concret, analitzar la transició de fase respecte de dues propietats. La primera que el graf percolar sigui connex i la segona que al graf percolat totes les components connexes siguin complexes. Entenent que una component connexa no és complexa si és un arbre o un graf unicíclic. D'altra banda, hem aplicat diferents models de grafs aleatoris per dur a terme aquest estudi experimental.

2. Concepte de graf

Per desenvolupar l'experiment, hem decidit escriure el nostre codi en c++ majoritàriament a excepció de dos programes, escrits en Python. A continuació, expliquem les diferents funcionalitats que conté el nostre programa.

2.1. Representació d'un graf

Per desenvolupar aquesta pràctica, vam decidir fer una classe graf amb la qual ens fos fàcil operar i cridar a les diferents funcions que necessitem per l'estudi a desenvolupar.

Nosaltres primerament vam decidir representar un graf amb llistes d'adjacència. Això ens funcionava molt bé fins que va arribar el punt on havíem de percolar de diferents maneres certs grafs. Ens vam adonar compte que amb grafs no molt grans, uns 1000 nodes, percolar arestes anava bastant lent. Això es devia que nosaltres per eliminar una aresta ho fèiem amb cost $O(m)$ on m és el nombre d'arestes d'un vèrtex qualsevol. Per tal d'aconseguir un algorisme més eficient, se'ns va ocórrer una manera de fer-ho en temps $O(1)$. Per fer-ho vam haver de sacrificar una mica d'espai, representant els grafs amb una matriu d'adjacència la qual fa que el graf ocupi $O(n^2)$. Un cop fet el canvi aquest, tant percolar nodes com arestes ho fem en temps $O(1)$, el que millora notablement l'algorisme. Hem obtingut en temps $O(1)$ tant en la percolació de vèrtex com d'arestes, ja que en comptes d'eliminar-los de memòria, els invalidem. A causa d'aquest canvi, a posteriori, indiferentment de l'operació, hem de comprovar si l'aresta i/o vèrtex existeixen, és a dir, si són vàlids.

Aquesta classe s'ocupa de totes les operacions relacionades amb grafs, per això hem hagut d'implementar el càlcul de components connexes, comprovar si un graf és unicíclic i comprovar si un graf és un arbre. Per fer tots tres algoritmes hem usat una modificació de l'algorisme del DFS.

Tant mirar si un graf és arbre com si és unicíclic utilitzem un algorisme molt semblant al DFS amb alguna petita variació, per tant, el cost d'aquests algorismes és de $O(V + E)$.

Com que no només volem calcular el nombre de components connexes, sinó que també guardar-les per treure resultats posteriorment, l'algorisme "s'encareix" una mica i finalment ens deixa amb un cost $O(V^2 + E)$. Això és degut al fet que emmagatzemem cada component connexa en forma de graf, per tant, després de fer el DFS, creem un graf en temps $O(V)$.

PSEUDOCODI DEL CÀLCUL DE COMPONENTS CONNEXES:

$for\ v\ \in\ V$:

    $if\ VertexExists(v)\ and\ not\ visited(v)$:

        $visitedDFS(n,\ false)$

        $dfs(v,\ visitedDFS)$

        $ConnectedComponent(n)$

        $for\ v'\ \in\ visitedDFS$:

            $ConnectedComponent[v'].pushback(graph[v'])$

Per veure si un graf és unicíclic, ens hem basat en el següent codi: (**https://www.geeksforgeeks.org/print-all-the-cycles-in-an-undirected-graph/**). Com deiem abans, és una adaptació del DFS.

2.2. Creació de grafs

Per la creació de grafs, hem creat les següents funcions:

- *Binomial random graph* que donat un nombre de nodes i una probabilitat, retorna un graph on cada aresta entre tots els nodes existents, té la probabilitat donada de ser o no al graf, per obtenir la probabilitat entre 0 i 1, simplement fem una crida a la

funció rand() i la dividim per RAND_MAX. Donat que hem de comprovar aquesta probabilitat per cada parell de nodes del graf, el cost el $O(n^2)$.

- R*andom geometric graph*, el que fem és donat un nombre de nodes i una distància r que va de 0 a 1, posicionem cada node dins d'un espai en 2D entre valors, on cada eix va de 0 a 1, per cada node posicionat dins l'espai comprovem la distància que es troba, si la distància és inferior o igual a r llavors existeix una aresta entre els nodes. Igual que la funció anterior, per donar un valor als eixos utilitzem la funció rand() i la dividim per RAND_MAX. L'algorisme té cost $O(n^2)$.

- *Mesh graph generator*, és un graf que donada una n, genera un graf de n x n vèrtexs, on tot vèrtex està connectat amb els seus veïns horitzontals i verticals consecutius a ell mateix. $O(n^2)$.

2.3. Percolació de grafs

Per la percolació de grafs, hem creat les següents funcions:

- Percolació de nodes: donat un graf i una probabilitat p, realitza una percolació dels nodes amb una probabilitat de p. $O(n)$.

- Percolació d'arestes: donat un graf i una probabilitat p, realitza una percolació de les arestes amb una probabilitat de p. $O(n^2)$.

3. Execució i recol·lecció de dades

Dins de la carpeta on es troba el codi podeu trobar un fitxer readme.md on es pot trobar una descripció del contingut de cada fitxer i com realitzar l'execució del programa.

Per a poder obtenir els resultats hem fet servir dos metodologies. La primera per a l'estudi de la família dels binomial random graph i per a la de random geometric graph; la segona només ha sigut feta servir per als grafs graella.

La primera metodología consisteix a:

1. Decidir la quantitat de vèrtexs els quals volem fixar per a la realització de tot l'estudi, en el nostre cas vam fer l'execució per a una n = {20, 60, 100, 1000, 5000}
2. Decidir cada quan voldríem obtenir uns resultats. Al principi vam probar a realitzar l'estudi amb un interval de [0, 1] i anar extraient els resultats cada 0.01. Això ens donaven uns resultats correctes però a l'hora de visualitzar les dades, vam trobar molt soroll a causa de la granularitat de l'obtenció de les dades. Vam decidir, per això, vam augmentar fins a 0.05 per tal de reduir el soroll generat per l'extracció de les dades.

3. Decidir quants grafs volem generar per a cada cop que volem extreure les dades. Per a tenir una mostra suficientment significativa, vam decidir realitzar 100 grafs cada vegada.

La segona metodología és semblant a la primera:

1. Decidim la n del graf graella. Tenim en compte que per a un n = 10, estaríem parlant d'un graf de 100 nodes, ja que la proporció de nodes respecte a la n és de n^2. Llavors per a aquest apartat vam decidir fer la selecció de n = {10, 20, 50, 60, 100}, que seria l'equivalent a 100, 400, 2500, 3600, 10000 nodes per graf.
2. Aquest pas serà el mateix que a l'apartat anterior però, aquests valors de l'interval [0, 1] els farem servir per a realitzar la percolació del graf i comprovar les propietats a estudiar.
3. En aquest cas, com el graf sempre serà el mateix al tenir el mateix número de nodes i les arestes no varien per la manera en la que es construeix, el número de vegades que farem l'experiment, servirà per veure quants cops realitzarem la percolació sobre el graf graella. Com a l'apartat anterior, vam decidir realitzar 100 percolacions.

Els resultats que volem obtenir són: comprobar per a cada p(el paràmetre que definim al pas 2), dels 100 grafs que generem per a cada p, comprobar quins d'aquests són connexos i quants d'aquests totes les seves components connexes són complexes. El resultat els obtenim fent una divisió de $grafsConnexs/100$ en el cas de la comprovació de la connexitivitat i $grafTotesCCComplexes/100$ en cas de trobar tots els grafs que totes les seves components connexes són complexes.

Un cop definit el procediment, és el moment d'executar l'algorisme. Un problema amb el que ens vam trobar va ser la "falta" de memòria. La nostra idea inicial era generar tots els grafs seguits i emmagatzemar-los dins d'un vector. El problema era que generar 100 grafs per a cada interval de 0.05 dins de l'interval [0,1] ens provocaba haber d'emmagatzemar dins de la memòria $1/0.05 * 100 = 2000$ grafs de 5000 nodes amb les seves arestes.

Vam cambiar la manera generant els grafs en moment d'execució i només guardar-lo mentre estem calculant les dades a buscar. Com fem servir la generació de números pseudoaleatoris amb una mateixa llavor, l'execució dels programes amb les mateixes entrades que han estat definides prèviament a les metodologies, donaran sempre els mateixos resultats.

Amb els paràmetres anteriors, el funcionament és bastant ràpid fins arribar als 5000 nodes. Amb aquesta quantitat de nodes i els paràmetres seleccionats per l'estudi, l'execució del programa podría arribar a trigar 1 hora aproximadament. Tenint en compte que estem parlant de 2000 grafs de 5000 nodes cadascun amb un màxim de 12000000 d'arestes per graf, és un temps raonable.

Un cop executat el programa en C++, obtenim per el canal de sortida estàndard alguna cosa semblant a la següent.

```
Introduce nodos:
20
Introduce probabilidad binomial/geometric:
0.05
Numero de grafos por probabilidad:
100
Empiezo a calcular
0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95,
0,0,0.05,0.46,0.75,0.95,0.98,1,1,1,1,1,1,1,1,1,1,1,1,
1,0.95,0.34,0.01,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

Sota la línia “Empiezo a calcula” tenim el següent:

- La primera fila consisteix en les probabilitats amb les quals hem realitzat l’obtenció de dades.
- La segona fila correspon a la proporció de grafs connexos en relació els grafs creats.
- La tercera fila correspon a la proporció de grafs els quals totes les seves components connexes són complexes.

Un cop obtingut això fent servir la llibreria de python Matplotlib, hem graficat els resultats per tal de tenir una interpretació visible de les propietats del graf. Amb els resultats obtindríem una forma similar a la següent:

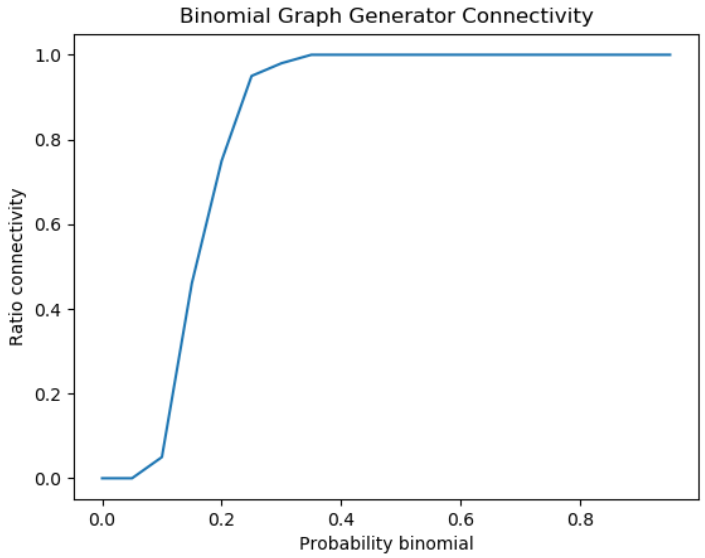

Figura 3.1 - Exemple de com queda un resultat graficat

Per tal de no omplir el document amb imatges de cada graf, hem decidit utilitzar una gràfica per tal de visualitzar tots els grafs generats de la mateixa família que estudien la mateixa propietat.

4. Resultats i conclusions

4.1 Connectivitat

En l'enunciat de la pràctica, se'ns donava una gràfica com a exemple del que ens hauria de sortir com a resultat, la gràfica en qüestió és la de la Figura 4.1. Per comprovar que realment la sortida fos correcte i d'aquesta manera comprovar la veracitat dels nostres càlculs, vam decidir generar una gràfica tenint en compte les mateixes variables, com es pot observar a la Figura 4.2.

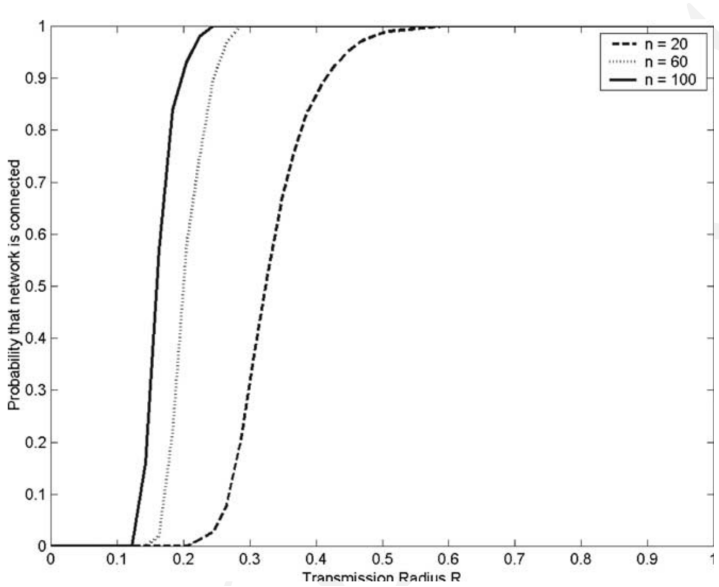

Figura 4.1 - Resultat previst

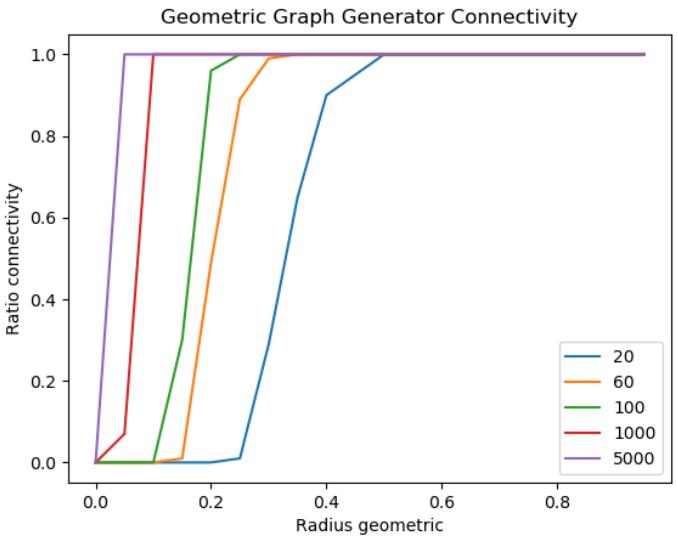

Figura 4.2 - Resultat obtingut amb el geometric graph generator

Com podem veure a la Figura 4.2, un graf assoleix la seva fase de transició més aviat com major és el nombre de nodes del graf. Això és perquè al haver-hi un major nombre de vèrtexs, és més probable que el nombre d'arestes augmenti, ja que els nodes es distribueixen per l'espai fent que existeixin parells de nodes prou propers perquè la distància entre aquests sigui menor o igual al radi, de manera que per dos nodes que abans no hi havia un camí entre ells, ara existeixi un node entre aquests dos que fa possible aquest camí, això aplicat a tot el graf fa més probable complir la propietat de connectivitat.

Mostrarem els resultats obtinguts de la fase de transició d'un graf geomètric fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\mathbf{\Pi}_{20} \in (0.45,\ 0.50)$
- $\mathbf{\Pi}_{60} \in (0.30,\ 0.35)$
- $\mathbf{\Pi}_{100} \in (0.20,\ 0.25)$
- $\mathbf{\Pi}_{1000} \in (0.05,\ 0.10)$
- $\mathbf{\Pi}_{5000} \in (0.00,\ 0.05)$

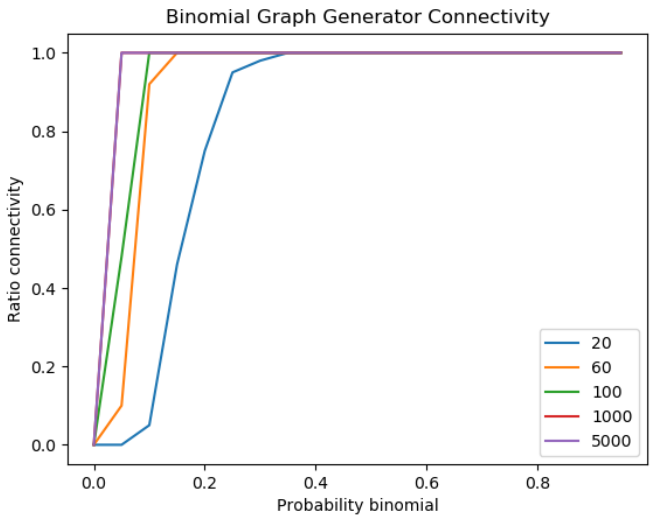

Figura 4.3 - Resultat obtingut amb el binomial graph generator

A la figura 4.3, podem observar el mateix fenomen que a la figura 4.2, amb una major quantitat de nodes, la fase de transició apareix d'una manera més ràpida. Això bé donat per l'explicació de ***binomial geometric graph***. Aquesta tècnica consisteix a visitar totes les possibles arestes dins d'un graf de n nodes i afegir aquella aresta amb una probabilitat *p*. Tenint en compte que el nombre màxim d'arestes d'un graf no dirigit és $n(n-1)/2$ això provoca que un graf amb moltes arestes disponibles per ser seleccionades augmenta la probabilitat de què aquest sigui connex; en canvi, un que té poques arestes, fàcilment podria donar-se el cas que cap aresta sigui seleccionada. La probabilitat de no seleccionar cap aresta ve donada per la fórmula $(1-p)^{n(n-1)/2}$, ja que estaríem parlant d'una successió de probabilitats encadenades. Amb una n molt gran i una p molt petita, serà quasi impossible que no seleccionem alguna aresta (estaríem parlant que el seu límit quan n

tendeix a infinit i la p és prou petita diferent de 0, donaria com a resultat 0); altrament amb un graf petit, la probabilitat seria diferent de 0.

Mostrarem els resultats obtinguts de la fase de transició d'un graf binomial fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\mathbf{\Pi}_{20} \in (0.30,\ 0.35)$
- $\mathbf{\Pi}_{60} \in (0.10,\ 0.15)$
- $\mathbf{\Pi}_{100} \in (0.10,\ 0.15)$
- $\mathbf{\Pi}_{1000} \in (0.0,\ 0.05)$
- $\mathbf{\Pi}_{5000} \in (0.0,\ 0.05)$

La família de grafs graella tenen un comportament similar als del binomial, amb una petita diferència. Primer generem un graf graella amb totes les arestes necessàries perquè compleixi la condició que pertanyi a aquella família i després li aplicarem una percolació amb una probabilitat p que decantarà si aquella aresta es manté al graf o no. La quantitat d'arestes d'un graf d'aquesta família és $n(n+1)$.

Parlant de la percolació d'arestes, com podem veure a la figura 4.4, un graf amb una quantitat més petita de nodes, aconsegueix més aviat una connectivitat més abundant que un de més nodes. Això ve definit per la següent expressió de probabilitats encadenades $p^{n(n+1)}$. Amb una p prou gran diferent de 1 i una n arbitràriament gran, el fet que existeixin totes les arestes és propera a 0 (parlem de què existeixin totes les arestes per tal d'intentar assegurar que el graf serà connex segur, encara que no estrictament necessari que existeixin totes les arestes del graf perquè aquest sigui connex); en canvi, en un graf amb una quantitat més petita de nodes, la probabilitat es transforma en una més gran.

Mostrarem els resultats obtinguts de la fase de transició d'un graf graella aplicant la percolació per arestes, fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\mathbf{\Pi}_{10x10} \in (0.90,\ 0.95)$
- $\mathbf{\Pi}_{20x20} \in (0.80,\ 0.85)$
- $\mathbf{\Pi}_{50x50} \in (0.85,\ 0.90)$
- $\mathbf{\Pi}_{60x60} \in (0.85,\ 0.90)$
- $\mathbf{\Pi}_{100x100} \in (0.90,\ 0.95)$

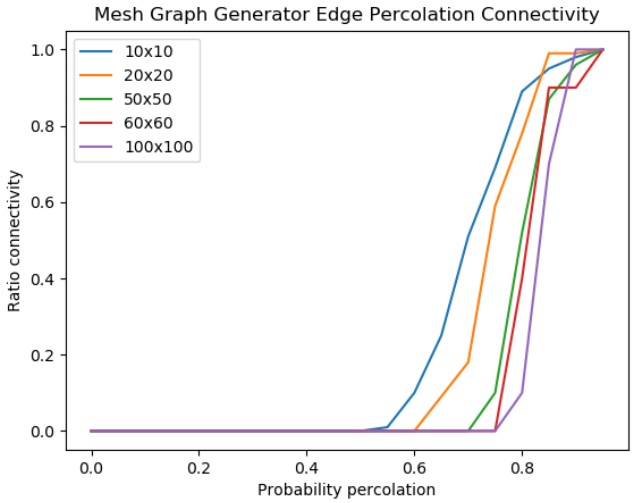

Figura 4.4 - Resultat obtingut percolant els nodes del graf Graella

En el cas de la percolació per vèrtexs, com podem veure a la figura 4.5, podem treure la mateixa conclusió que a l'apartat anterior; quants menys nodes té, més fàcil el graf sigui connex per la mateixa raó p^n. Per a una p gran diferent de 1 i una n prou gran, fa que el límit d'aquesta funció tendeix a 0. En canvi, amb un n petita, fa que l'operació arribi a ser propera a 1 o com a mínim diferent de 0.

Mostrarem els resultats obtinguts de la fase de transició d'un graf graella aplicant la percolació per vèrtexs, fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\mathbf{\Pi}_{10x10} \in (0.85,\ 0.90)$
- $\mathbf{\Pi}_{20x20} \in (0.90,\ 0.95)$
- $\mathbf{\Pi}_{50x50} \in (0.95,\ 1.00)$
- $\mathbf{\Pi}_{60x60} \in (0.95,\ 1.00)$
- $\mathbf{\Pi}_{100x100} \in (0.95,\ 1.00)$

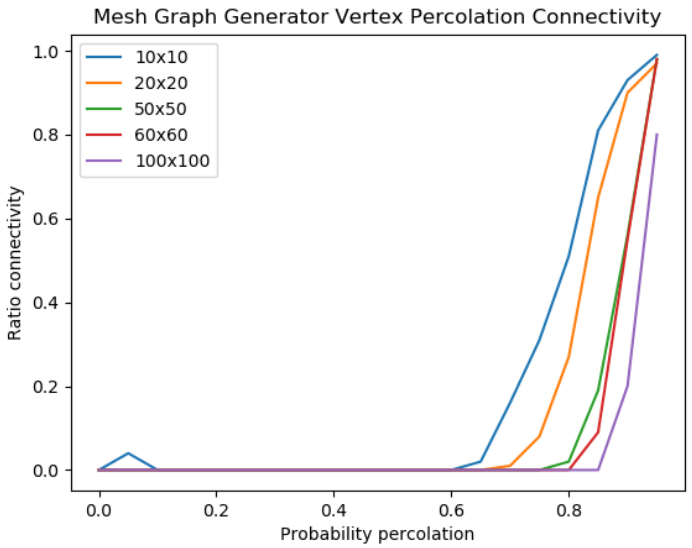

Figura 4.5 - Resultat obtingut percolant les arestes del graf Graella

4.2 Complexitat

Les següents gràfiques ens mostraran com es comporta la propietat de complexitat amb diferents models de creació de grafs.

El primer model presentat és el de “geometric graph generator”. Com podem veure en la Figura 4.6.

Notem, es comporta de manera inversa a la connectivitat estudiada anteriorment, en aquest cas, quan el radi geomètric sigui més gran, la probabilitat que formi un graf complex serà més difícil, ja que l'aparició d’arestes entre vèrtexs serà molt gran i farà que puguin aparèixer més d’un camí per qualsevol parell de vèrtexs o més d’un cicle per a un mateix graf, fet que provocarà que aquella component connexa no sigui complexa. Menys el graf de 20 nodes, els altres tenen una descendència a la gràfica bastant similar, fet que ens ha dificultat molt poder acotar un valor de transició per a la propietat de complexitat.

Mostrarem els resultats obtinguts de la fase de transició d’un graf geomètric, fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\boldsymbol{\Pi}_{20} \in (0.00,\ 0.01)$
- $\boldsymbol{\Pi}_{60} \in (0.00,\ 0.05)$
- $\boldsymbol{\Pi}_{100} \in (0.00,\ 0.05)$
- $\boldsymbol{\Pi}_{1000} \in (0.00,\ 0.05)$
- $\boldsymbol{\Pi}_{5000} \in (0.00,\ 0.05)$

Per a la creació de grafs binomial, tenim el resultat a la figura 4.6. L’explicació és semblant a l’anterior; per a una probabilitat més gran, és més probable l’aparició de multicicles dins

d'una component connexa per la definició de construcció d'aquest graf a partir de la p desitjada. Vam trobar el mateix problema que a l'apartat anterior, se'ns fa molt difícil poder donar una xifra que ens doni la fase de transició de la propietat de complexitat.

Mostrarem els resultats obtinguts de la fase de transició d'un graf binomial, fent servir uns intervals, ja que no podem assegurar exactament quina és la fase de transició *(+-0.05)*:

- $\mathbf{\Pi}_{20} \in (0.05,\ 0.10)$
- $\mathbf{\Pi}_{60} \in (0.00,\ 0.05)$
- $\mathbf{\Pi}_{100} \in (0.00,\ 0.05)$
- $\mathbf{\Pi}_{1000} \in (0.00,\ 0.05)$
- $\mathbf{\Pi}_{5000} \in (0.00,\ 0.05)$

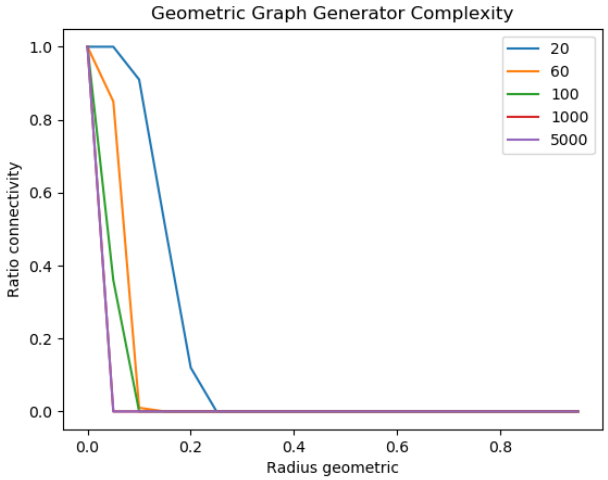

Figura 4.6 - Resultat obtingut amb el geometric graph generator

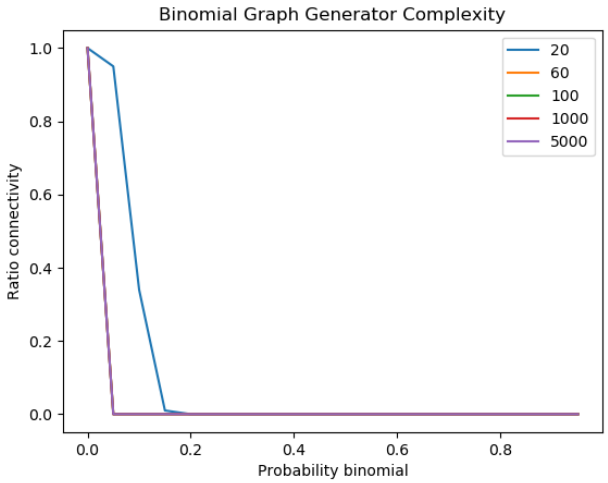

Figura 4.7 - Resultat obtingut amb el binomial graph generator

Per últim, la família dels grafs graella ens ha donat mostrat uns resultats per a nosaltres sorprenents, ja que no vam trobar cap graf que fos complex. Hem tret dues conclusions:

- De la part de percolació de vèrtexs, quan treiem un vèrtex, automàticament estem eliminant totes les arestes a les quals aquell vèrtex pertany fet que provoca que es puguin arribar moltes components connexes. Com cada vèrtex té assignades com a molt 4 arestes i com a poc 2, el fet de mantenir-les provoca que fàcilment aquelles arestes passin a formar part d'un cicle de la component connexa a la qual pertanyen, fet que fa que puguin aparèixer múltiples cicles.

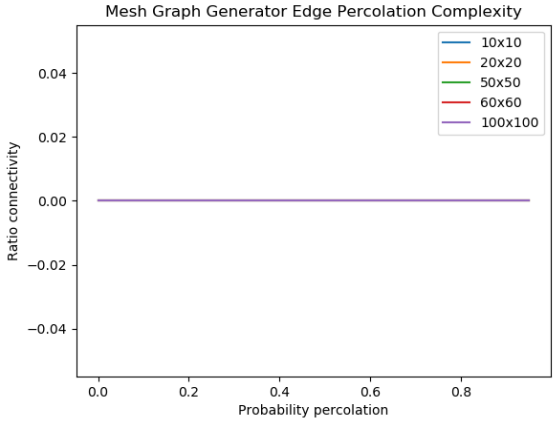

Figura 4.8 - Resultat obtingut percolant les arestes del graf Graella

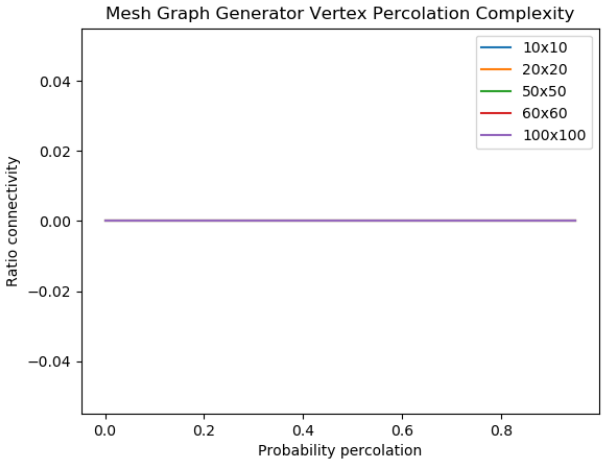

Figura 4.9 - Resultat obtingut percolant els vèrtex del graf Graella