# Transició de fase en propietats de grafs

GRAU - A
CURS 2018 - 2019

**Guillem Pla Bertran**
**Arnau Ruana Ramos**
**Francesc Ruiz Tuya**

# Taula de Continguts

# Introducció

El nostre objectiu en aquest treball és estudiar la transició de fase per diferents propietats de grafs. Aquestes propietats cal estudiar-les després d'haver percolat un graf. Els grafs a percolar han de ser aleatoris segons un mateix model (definits en la secció *Models de graf utilitzats*) i la propietat d'aquest a estudiar ha de ser una de les definides a la secció *Propietats dels grafs estudiades* del treball.

Entenem que un graf ha estat percolat si ha passat per un procés de percolació. Aquest procés es pot fer de dues maneres diferents, per vèrtexs o per arestes. La percolació per vèrtexs consisteix en decidir per cada vèrtex d'un graf si aquest forma part del nou graf percolat o no. Similarment decidim si cada aresta d'un graf hi forma part o no. Per a decidir si un vèrtex o aresta forma part del nou graf ho farem amb una probabilitat $q$, que la definirem com la probabilitat de fallida que té un vèrtex o aresta d'un graf.

La transició de fase que hem d'estudiar per als diferents grafs consisteix en obtenir en quin valor de $q$ (probabilitat de percolació d'un component del graf) el graf aleatori passa de complir la propietat estudiada a no complir-la per a la majoria dels casos.

# Generador de grafs

En aquest projecte hem decidit utilitzar la llibreria *NetworkX* de *Python* per a la generació de grafs. Aquesta llibreria ens permet generar els grafs que necessitem durant el projecte, ja sigui una graella quadrada o un model de graf aleatori. Un cop feta la generació del graf el tractem per a que el programa en Python doni la sortida desitjada.

Hem decidit que la sortida sigui un enter *N*, que marca el nombre de vèrtexs d'un graf *G*, i el seguidament totes les arestes *E* del mateix graf. Les arestes estan representades per dos enters que representen els dos vèrtexs connectats. Aquesta sortida es produeix en un fitxer text que ens servirà d'entrada del programa principal. D'aquesta manera aquest podrà llegir el graf generat i el podrà processar.

Per aquest projecte hem fet servir:

`grid_2d_graph(N,N)` per a generar un graf graella quadrada, on N és l'arrel quadrada del nombre de vèrtexs.

`barabasi_albert_graph(N, M, seed)` per a generar un graf de tipus Barabasi-Albert, on N és el nombre de vèrtexs del graf, M el nombre d'arestes a afegir a un node i seed la llavor en que es genera la aleatorietat.

`watts_strogatz_graph(N, K, P, seed)` per a generar un graf Watts-Strogatz, on N és el nombre de vèrtexs del graf, K la llunyania en el que un node pot estar connectat amb un altre, P la probabilitat de que una aresta canviï i seed la llavor en que es genera la aleatorietat.

# Estructures de dades i algorismes

A continuació explicarem detalladament les estructures de dades i algorismes que fem servir en el nostre projecte. Especificarem què fa cada una de les funcions més rellevants de cada classe i quin cost en el temps tenen.

Els codis font corresponents es poden trobar al següent path: *src/*

## Union-Find

Aquesta estructura de dades està definida com a classe als fitxers font *unionfind.hh* i *unionfind.cc*. Es tracta d'una estructura de dades adient per mantenir en tot moment els nodes d'un graf agrupats per components connexes, la qual cosa ens permetrà comprovar si es compleixen les propietats estudiades d'una forma fàcil i eficient, un cop format el union find corresponent.

En teoria un DFS és més ràpid que un union-find perquè es pot garantitzar un cost constant, en canvi el union-find no. Però a la pràctica és més ràpid perquè no necessita crear una representació completa del graf.

A continuació definim amb més detall els atributs i mètodes que hem utilitzat per a aquesta estructura de dades.

### Descripció d'atributs

En el nostre Union-Find hem decidit posar-hi tres atributs: un enter n i dos vectors de enters *id* i *sz*.

L'atribut *n* és la mida dels vectors *id* i *sz*, que es calcula com a el nombre de vèrtex dels grafs més dos, que corresponen als vèrtexs virtuals *top* i *bottom*. Aquests ens serveixen per a comprovar si existeix un camí des d'un conjunt de vèrtex d'entrada a un conjunt de sortida.

Cada posició de l'atribut *id* representa un vèrtex, per a cada vèrtex conté l'identificador del pare. Amb aquest vector podem mantenir una estructura d'arbre on cada vèrtex d'un graf que estigui connectat amb un altre comparteixen la mateixa arrel. L'arrel és un vèrtex amb un identificador igual a la seva posició en el vector *id*. El pare és un vèrtex que té altres vèrtex que tenen el seu identificador, no cal que sigui l'arrel ja que pot tenir un pare.

Cada posició de l'atribut *sz* representa un vèrtex, per cada vèrtex conté el nombre de fills més 1 que té. Ens serveix per a poder unir dos arbres de forma que quedi el mínim arbre possible.

Descripció de mètodes

`int find(int i);`
Retorna l'arrel del vèrtex *i*. Com que estem tractant amb arbres el cost de *find()* serà de **O(h)**, essent *h* l'alçada de l'arbre. Per tant hem d'intentar que *h* sigui el menor possible. La implementació que hem escollit fa que en el moment de buscar un element passi a apuntar al seu padrí (pare del seu pare) i es cerqui l'arrel d'aquest. Es repeteix el procés a cada iteració fins a trobar l'arrel.

`bool unify(const int p, const int q);`
Retorna cert si els vèrtexs *p* i *q* s'han unir. Es busca les arrels dels dos vèrtexs amb un *find()*, si són iguals els dos nodes pertanyen al mateix grup i per tant no els unim. Si són diferents unim el vèrtex amb menys fills amb el que en té més. Els cost d'aquest algorisme és **O(h)** que correspon a la crida *find()*.

`void initializeTopBottom(list<int>& top, list<int>& bottom);`
Uneix els elements *top*, que és una llista amb identificadors de vèrtexs, amb la posició *n-2* del vector *id* i els elements de la llista bottom amb el vèrtex virtual n-1. Aquest mètode ens permet assignar una llista de vèrtex d'entrada i sortida i per tant ens permet comprovar amb facilitat si una llista de vèrtexs connecta amb una altra després de percolar. El cost és **O(n*h)**, essent *n* el nombre d'elements màxim d'una de les dues llistes i *h* el cost corresponent a cada crida *unify()*.

`bool connected();`
Retorna cert si es compleix que el vèrtex virtual *n-1* està connectat amb *n-2*. Ens serveix per a comprovar si dos conjunts de vèrtexs, un d'entrada i un de sortida, estan connectats. El cost d'aquest mètode és **O(h)** corresponent a la crida *find()*.

## Graph

Aquesta estructura de dades està definida com a class als fitxers font *graph.hh* i *graph.cc*.

Hem considerat necessària aquesta classe perquè ens permet llegir, escriure, modificar i mantenir en memòria un graf. Consta d'un vector de llistes graph i dos enters *nEdges* i *nVertices*.

A continuació procedim a explicar més detalladament els atributs i mètodes de la classe *Graph*.

## Descripció d'atributs

L'atribut graph representa una llista d'adjacències, que és una de les formes per representar grafs més usada. Cada posició del vector representa un vèrtex i aquest conté una llista amb els seus vèrtexs adjacents. En cap moment hem de borrar cap vèrtex ja que hem considerat que un vèrtex amb una llista d'adjacència buida no existeix. Per tant podem utilitzar un vector, això ens permet accedir-les immediatament. Les arestes es representen en una llista perquè, en el procés de percolació, podem esborrar-les d'una manera fàcil. El cost en espai és de $O(|V|+|E|)$.

Els atributs nEdges i nVertices representen el nombre d'arestes i vèrtexs que té un graf respectivament després d'haver estat percolat.

## Descripció de mètodes

```
void addEdge(const int vert0, const int vert1);
```
Donats dos enters, que representen dos vèrtexs, afegeix l'aresta corresponent a la unió d'aquests dos vèrtexs. Es suma 2 a nEdges.

```
void deleteEdge(int vert0, int vert1);
```
Donats dos enters, que representen dos vèrtexs, suprimeix l'aresta corresponent a la unió d'aquests dos vèrtexs. Es resta 2 a nEdges.

```
void deleteVert(int vert);
```
Donat un enter, que representa un vèrtex, suprimeix totes les seves arestes adjacents.
Es resta el nombre d'arestes borrades a nEdges.

```
void read();
```
Llegeix pel canal d'entrada estàndard parelles d'enters que corresponen a una aresta i aquesta s'afegeix al graf.

```
Graph percolateVertices(int numerador, int denominador);
```
Funció encarregada de percolar els vèrtexs d'un graf segons un certa probabilitat de fallida passada com a fracció (numerador i denominador). En el pitjor dels casos per cada vèrtex pertanyent al graf haurem d'esborrar-lo, donat un vèrtex la funció *deleteVert()* esborra totes les seves arestes. Per tant donat un graf visitarem tots els vèrtexs i totes les arestes, el que resulta en un cost de $O(|V|+|E|)$.

`Graph percolateEdges(int numerador, int denominador);`
Funció encarregada de percolar les arestes d'un graf segons un certa probabilitat de fallida *q* passada com a fracció (numerador i denominador).
Si ens hi fixem detalladament obsarvem que enlloc d'eliminar les arestes segons la possibilitat *q* s'afegeixen segons la probabilitat complementaria. Aquest algorisme visita tots els vèrtexs i per cada vèrtex visita totes seves arestes, a cada iteració es crida a la funció *addEdge()* i *deleteEdge()* que tenen un cost O(1). Per tant en el pitjor dels casos recorrem tots els vèrtexs i totes les arestes una vegada, el que resulta en un cost de **O(|V|+|E|)**.

`bool checkConnected(list<int>& top, list<int>& bottom);`
Aquesta funció retorna cert si existeix un camí que recorre el graf des d'algun dels nodes *top* fins a algun dels *bottom*. Retorna fals altrament. En el pitjor dels casos recorrem tots els vèrtexs i totes les arestes una vegada. Per tant el cost en el temps és de **O(|V|+|E|)**.

`bool checkCicles();`
Funció encarregada de retornar si existeix algun cicle en el graf. En el pitjor dels casos no trobem un cicle, per tant visitem tots els vèrtexs i totes les arestes un cop. El cost en el temps és de **O(|V|+|E|)**.

`void print();`
Mètode encarregat d'escriure perl canal estàndard de sortida un graf en un determinat format.

# Programes principals

Tots els programes principals (que es poden trobar a la carpeta src) s'encarreguen essencialment de la realització de tres tasques ben diferenciades entre sí.

La primera d'elles consisteix en llegir el graf amb el qual volem realitzar els experiments en un format compatible amb els que hagi generat el fitxer python corresponent.

La segona part es dedica a processar el graf obtingut percolant-lo per vèrtexs i per arestes. A la vegada va generant internament la sortida del main segons els resultats parcials que va obtenint a cada un dels experiments per a cada un dels increments de $q$ (definits com a constants en la part superior dels main).

Finalment, l'última tasca consisteix en imprimir per pantalla totes les dades intermèdies que ha anat capturant sobre un determinat graf i propietat per a la seva posterior anàlisi.

# Models de graf utilitzats

Per a poder fer l'estudi experimental de la transició de fase hem utilitzat tres models de grafs diferents. Una graella quadrada, el graf aleatori Barabasi-Albert i el graf aleatori Watts-Strogatz. Els grafs aleatoris són aquells que han sigut generats per algun tipus de procés aleatori.

## Graella quadrada

El graf graella quadrada *NxN* és un tipus de graf el qual els seus vèrtexs corresponen als punts en el pla amb coordenades enteres. En el nostre cas tant les coordenades-y com les coordenades-x estan en el rang 0..N-1. Dos vèrtexs estan connectats per una aresta sempre i quan els corresponents punts es trobin a una distància de 1. Aquest tipus de graf forma un mosaic regular.

Algunes de les propietats d'aquest graf són les següents:

- El nombre d'arestes és $2(N^2+N)$ arestes.
- És un graf hamiltonià
- És un graf bipartit

## Barabasi-Albert

El model de graf Barabasi-Albert és un model que s'usa per a generar xarxes lliures d'escala. Aquestes són un tipus específic de xarxa complexa. En una xarxa lliure d'escala, alguns nodes estan altament connectats, encara que el grau de connexió de la majoria de nodes és baix. En la natura podem trobar aquest tipus de distribució (internet, xarxes socials, etc.). Segueixen la següent llei potencial:

$$P(k) = ck^{-\gamma}$$

La fracció *P(k)* de nodes en la xarxa té *k* connexions a la resta de nodes amb grans valors de k. L'exponent és un paràmetre que normalment està entre 2 i 3, encara que de vegades pot no estar-hi. *c* és una constant proporcional.

Aquest model inclou els conceptes de creixement i connectivitat preferent. El creixement vol dir que quan es genera el graf augmenta el nombre de vèrtexs amb el temps i la connectivitat preferent és la probabilitat que un node rebi més links. Llavors tenim que com més links tingui un node més probabilitats tindrà de rebre'n un altre.

## Watts–Strogatz

El model de graf Watts-Strogatz és un model de generació de grafs aleatoris que produeix grafs amb xarxes de món petit. Aquest consta de distàncies mitjanes petites i valors grans del coeficient d'agrupament.

Una xarxa de món petit és un tipus de graf per el que la majoria dels nodes no són veïns entre ells, però en canvi es pot arribar a la majoria de nodes desde qualsevol altre node en un nombre petit de salts.

El concepte de distàncies mitjanes petites es defineix com al nombre de passos per a recórrer el camí més curt d'entre totes les parelles possibles de nodes.
El coeficient d'agrupament és una mesura del grau en que els nodes d'un graf tendeixen a agrupar-se.

Per a generar un graf Watts-Strogatz cal establir una xarxa unidimensional inicial amb N nodes, amb aquests nodes es creen en forma d'anell. Després cada node en l'anell s'uneix amb el seu *k* veí més proper (o *k-1* si *k* és senar). Llavors s'afegeixen dreceres que reemplacen algunes arestes amb una probabilitat *p*.

# Propietats dels grafs estudiades

Per a poder realitzar l'últim apartat del treball, hem hagut d'experimentar la transició de fase per a diferents propietats de grafs (addicionalment a la propietat dels apartats anteriors: travessia). Aquestes propietats han estat estudiades per a diferents models de graf aleatoris definits en la secció anterior del treball.

Les propietats de grafs per a les quals hem decidit estudiar la transició de fase amb diferents tipus de graf són les següents:

## Travessia

Definim travessia com a l'existència d'un camí que recorre un graf percolat determinat des d'un conjunt de vèrtexs "entrada" fins a un conjunt de vèrtexs "sortida" predefinits amb anterioritat.

## Cicles

Aquesta segona propietat consisteix simplement en saber si un graf percolat determinat té algun cicle o, per contra, no en té cap i, per tant, és una arbre.

Entenem per cicle la successió d'arestes adjacents on es retorna al vèrtex de partida sense haver recorregut dos cops la mateixa aresta del graf.

# Anàlisi de les dades obtingudes

En aquest apartat inclourem algunes de les gràfiques obtingudes després de processar les dades per a cada tipus de graf i propietat estudiada i unes breus conclusions extretes. Ho farem amb les gràfiques que creiem més importants a l'hora d'argumentar els nostres resultats. No obstant, el document complet (amb totes les gràfiques generades) en format full de càlcul es pot trobar als paths següents: *doc/data*.ods*

Agruparem les gràfiques per tipus de graf i propietat estudiats. Dibuixarem primer les gràfiques (amb dues percolacions: vèrtexs i arestes) i, posteriorment, farem l'anàlisi del conjunt.

## Travessia + Graella quadrada

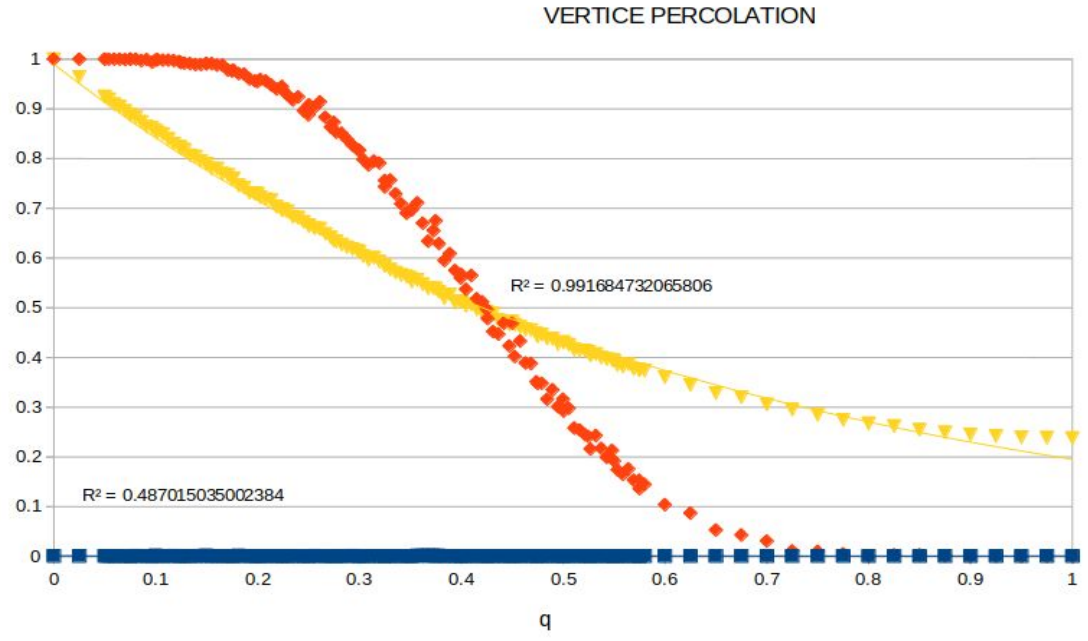
Per a realitzar els testos amb grafs Graella quadrats hem anat variant la mida en vèrtexs del generador pertinent en python.

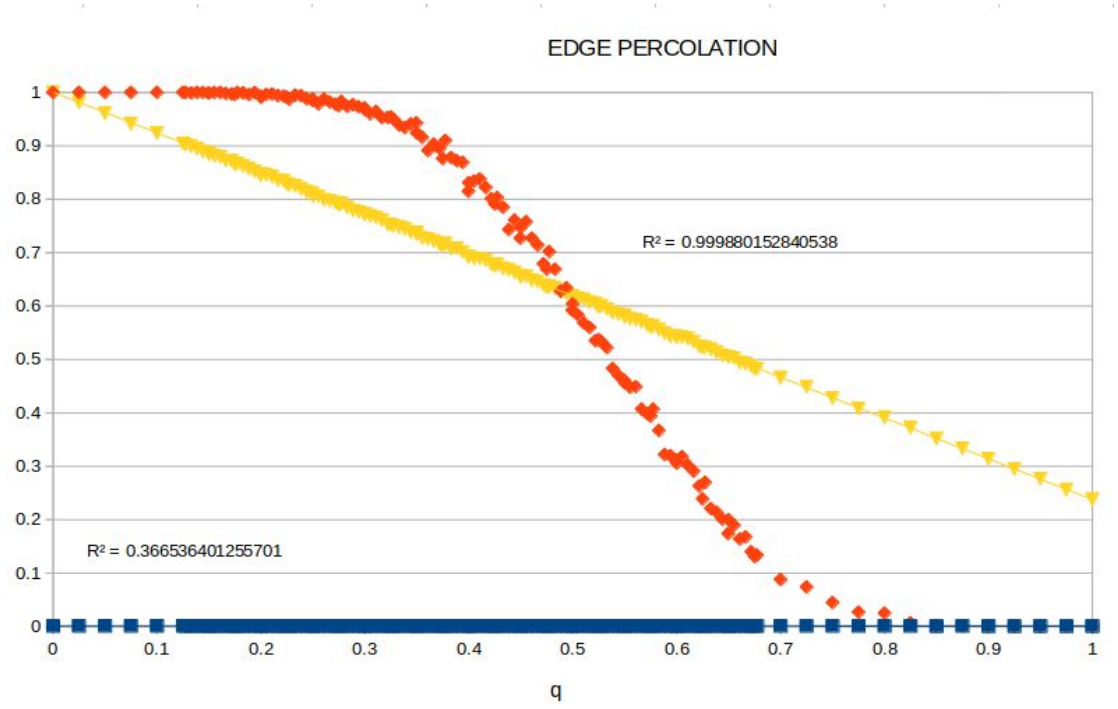
A continuació es mostren les gràfiques obtingudes per a diferents valors de *N* (nombre de vèrtexs). Les línies de color taronja representen el compliment de la propietat del graf i les grogues representen la suma de (V + E) respecte al valor màxim de vèrtexs i arestes que pot tenir el graf.

L'eix de les abscisses fa referència a la *q* i el de les ordenades al tant per u.

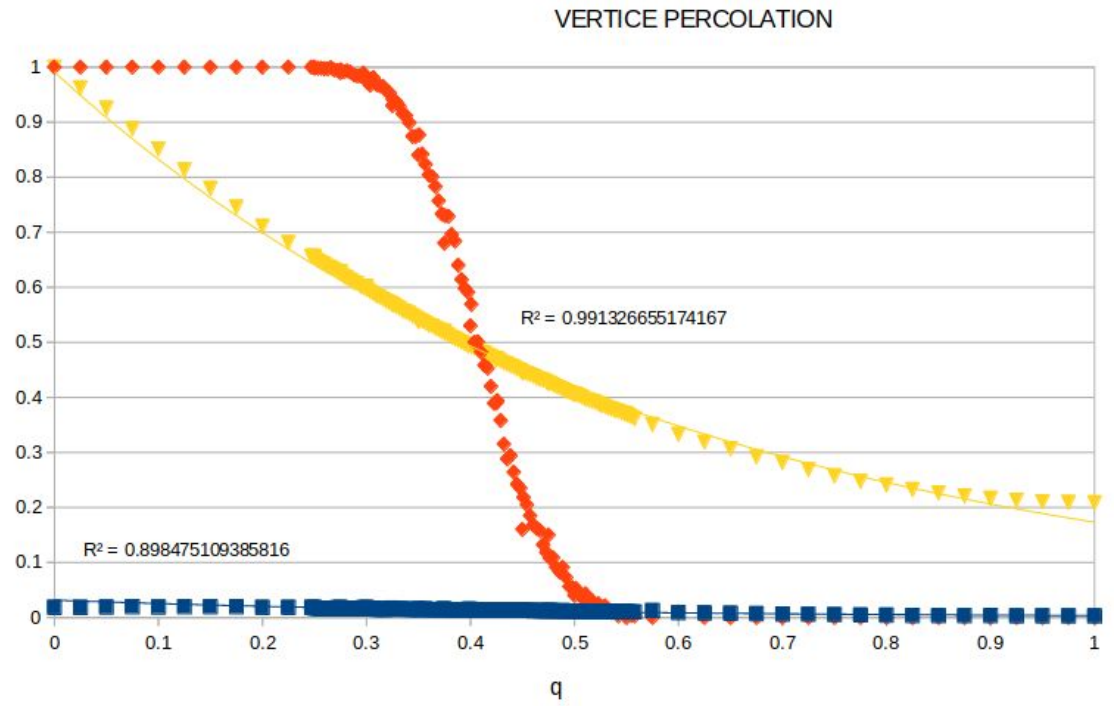
En aquelles gràfiques que apareix, la línia blava representa el temps en ms d'execució mitjana. Degut a que no vam poder demostrar la correlació entre el V+E i aquest temps, no la vam incloure a la resta de gràfiques.

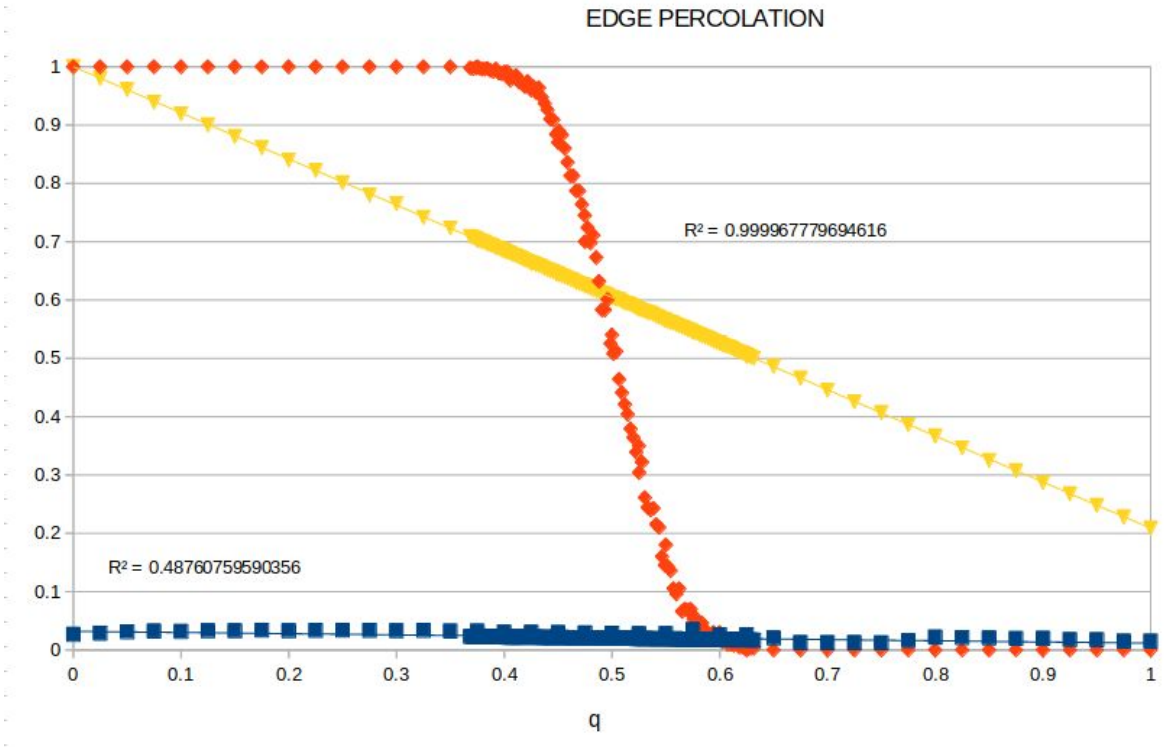
```
nombre de vèrtexs => 5^2
```



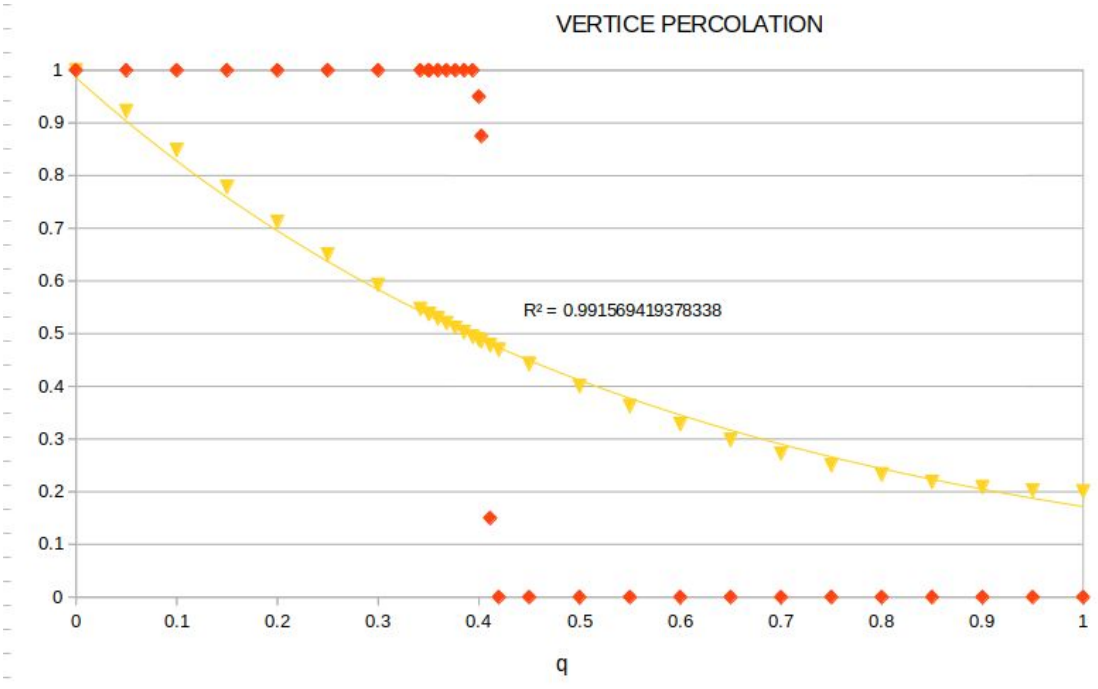


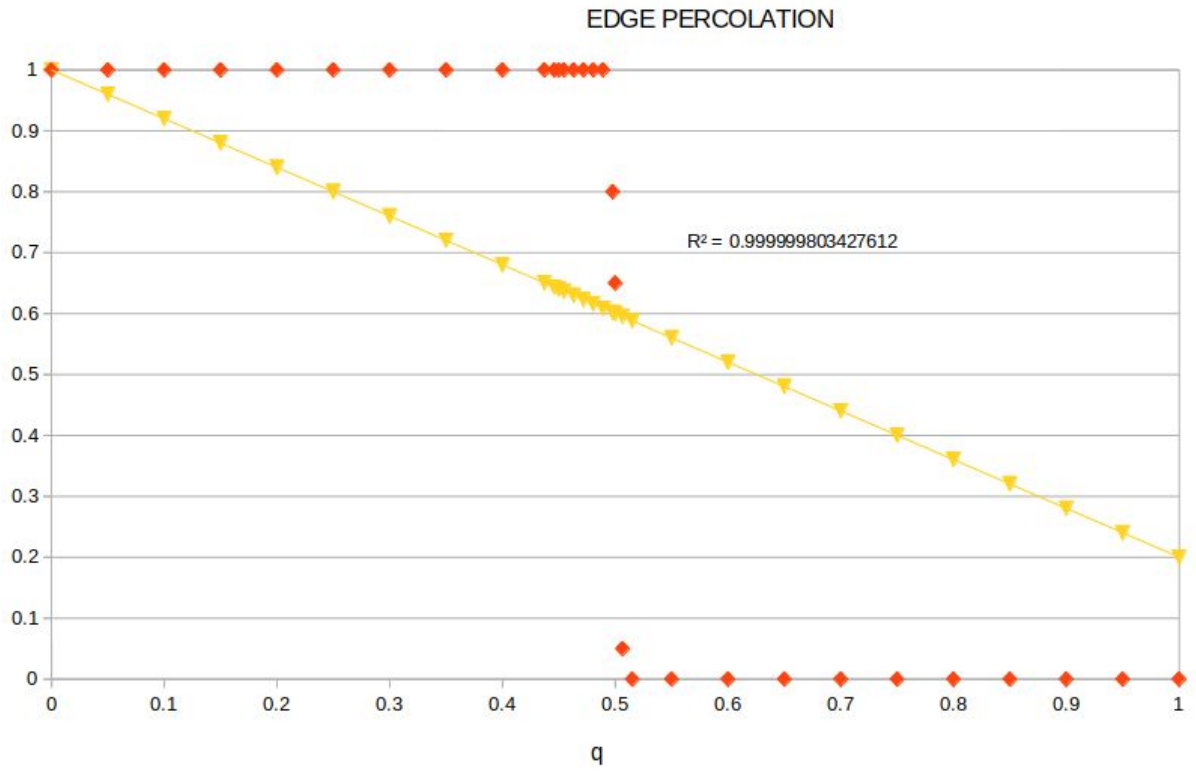

nombre de vèrtexs => 20^2





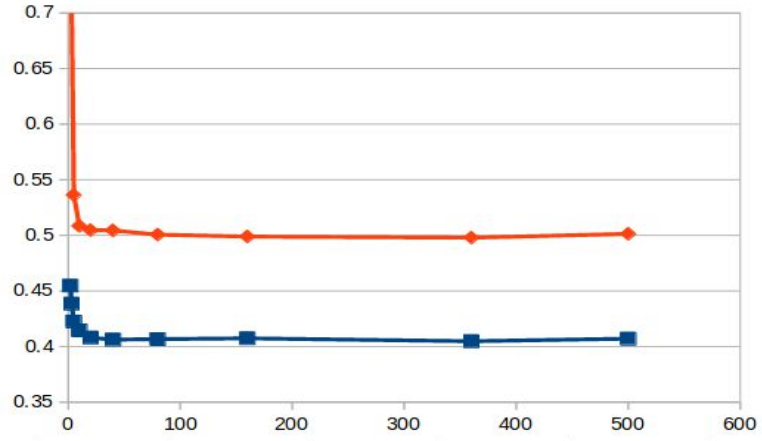

nombre de vèrtexs => 500^2



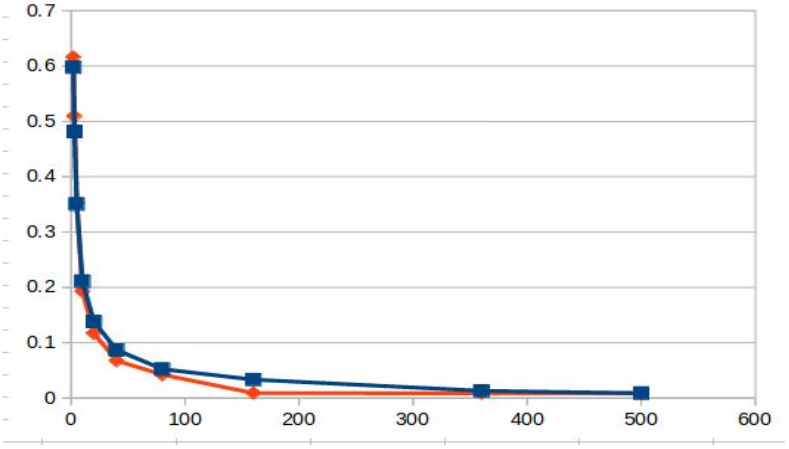



`resultat final`

La línia taronja representa la percolació per arestes i la blava per vèrtex.
L'eix X representa l'arrel del nombre total de vèrtex.



Eix Y:
valor de la *Q* que fa que la propietat travessa tingui una probabilitat del 50%. A aquesta *Q* específica ens hi referirem com a Q.5



Eix Y:
distància entre les *Q* amb propietat travessa 0,9 i 0,1

`conclusions`

Com que l'interval en el que es produeix la transició de fase tendeix a 0, el pendent amb què es produeix el canvi és cada cop més gran.

Per altra banda en el cas de la percolació per vèrtex, la transició de fase sempre es produirà amb una N inferior a la necessària en el casa de la percolació per aresta. En les dues percolacions quan el valor de N tendeix a infinit, Q.5 s'aproxima a un valor. En el cas de la percolació per vèrtex aquest valor és 0,4 i en el cas de la percolació per arestes el valor és 0,5.

El tant per u de V+E respecte a V+ E màxims disminueix d'acord amb dues regressions diferents. En el cas de la percolació per vèrtex, la disminució de V+ E obeeix a una regressió exponencial. En el cas de les arestes aquesta regressió és lineal. Els coeficients de regressió propers a u ens indiquen que les dues regressions són vàlides.

Tot i que el temps d'execució hauria d'estar relacionat amb el tamany de l'entrada, hem obtingut resultats contradictoris i hem optat per no extreure'n conclusions.

## Travessia + Barabasi-Albert

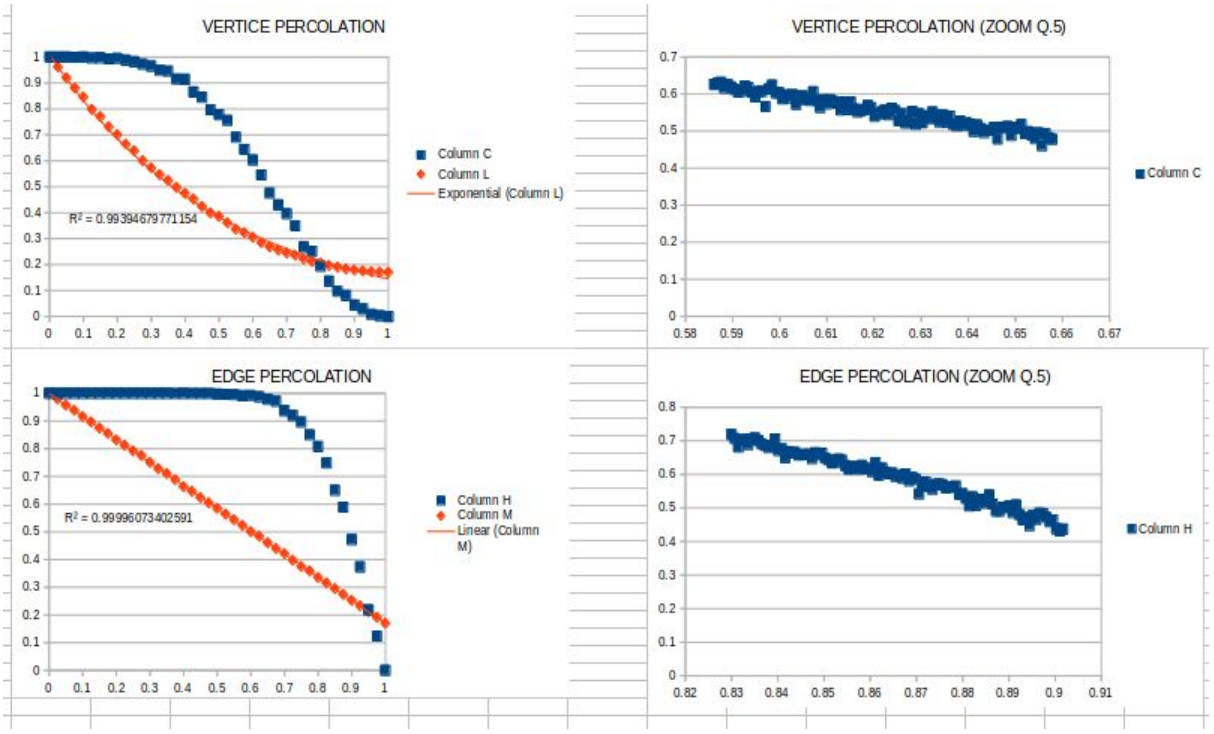
Per a realitzar els testos amb grafs Barabasi el generador de grafs en python ens obliga a afegir-hi un paràmetre extre *M*, a part del nombre de vèrtexs. Per no haver de fer estudis segons dues variables aleatòries ens hem limitat a mantenir constant el valor d'aquesta variable 'M' i anar variant únicament el valor dels vèrtexs.

Amb tot això present, obtenim les gràfiques que es veuen més endavant. En aquest cas les línies blaves representen el compliment de la propietat del graf i les vermelles representen la suma de (V + E) respecte al valor màxim de vèrtexs i arestes que pot tenir el graf (per exemple en el cas on $q = 0$ no hem eliminat cap aresta de manera que aquest coeficient serà 1).

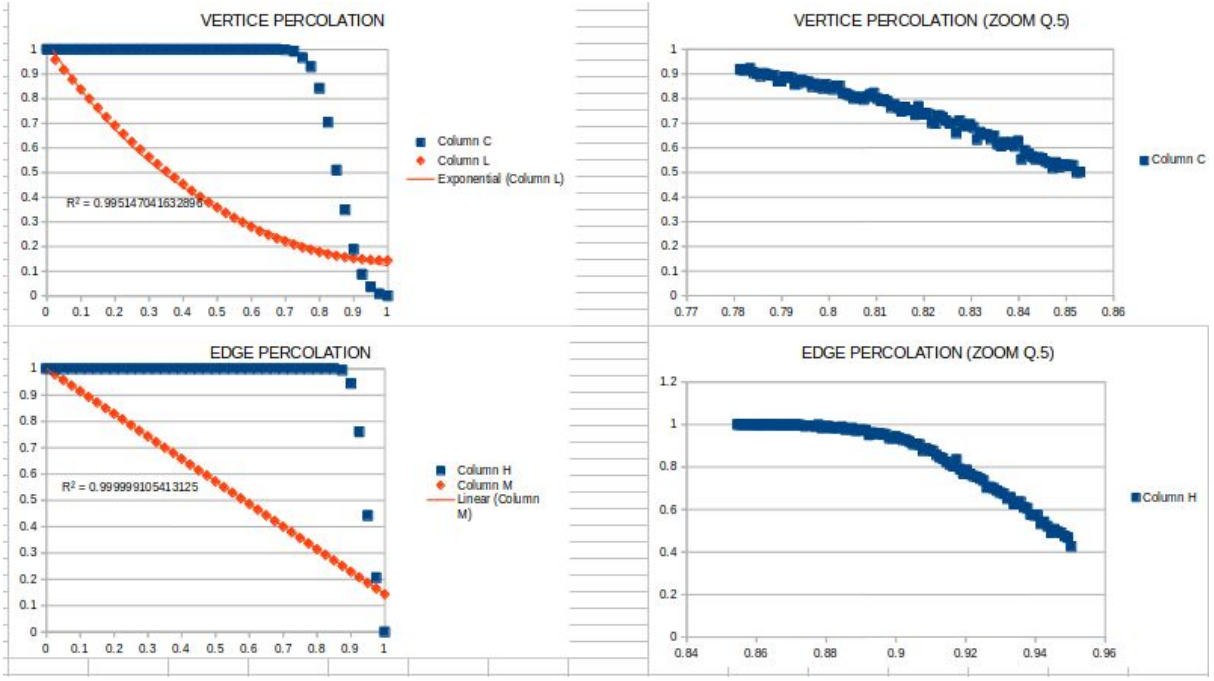
Els eixos són exactament els mateixos que en l'estudi dels graf graella quadrada, és a dir, l'eix X representa la probabilitat de fallida (q) i l'eix Y representa el tant per u.

A més, a diferència de l'apartat anterior, el zoom en valors crítics l'hem realitzat en una gràfica a part (a la dreta) per tal d'identificar millor la transició de fase en els punts més important i, alhora, conflictius per graficar.

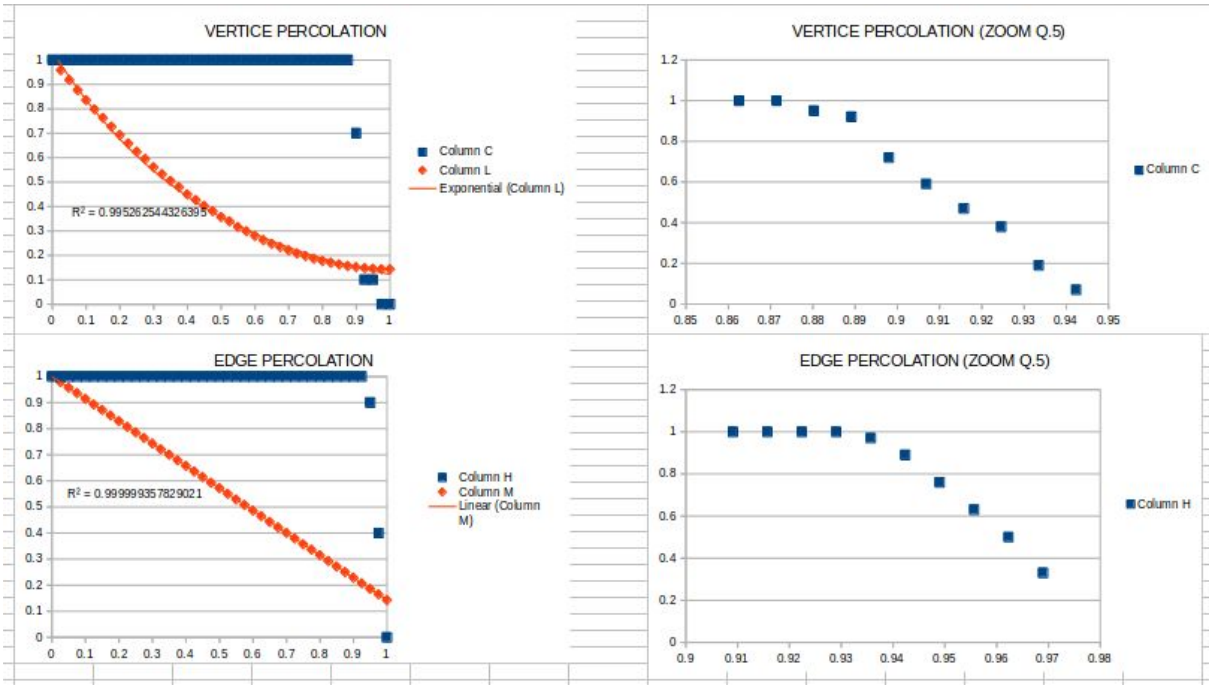
nombre de vèrtexs => 16




nombre de vèrtexs => 1024



`nombre de vèrtexs => 62500`



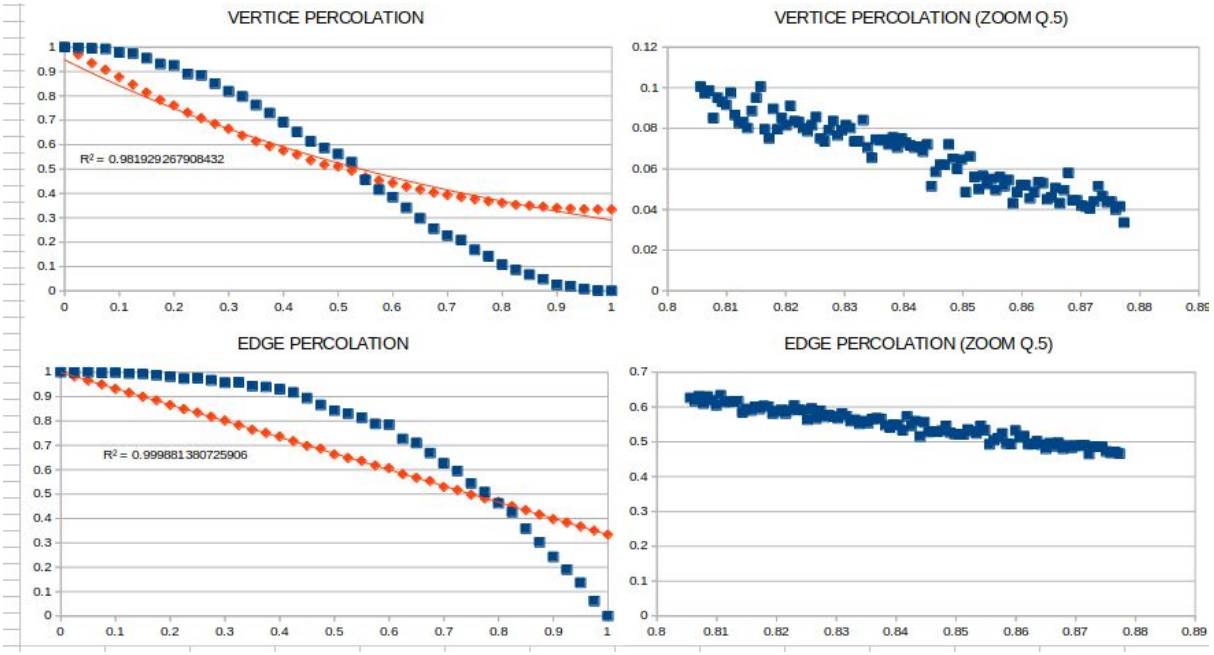

`conclusions`

Degut a que la disminució del nombre d'arestes és més ràpida en la percolació per vèrtex, la transició de fase per a la propietat travessa sempre es produeix amb una $q$ inferior a la de la percolació per arestes.

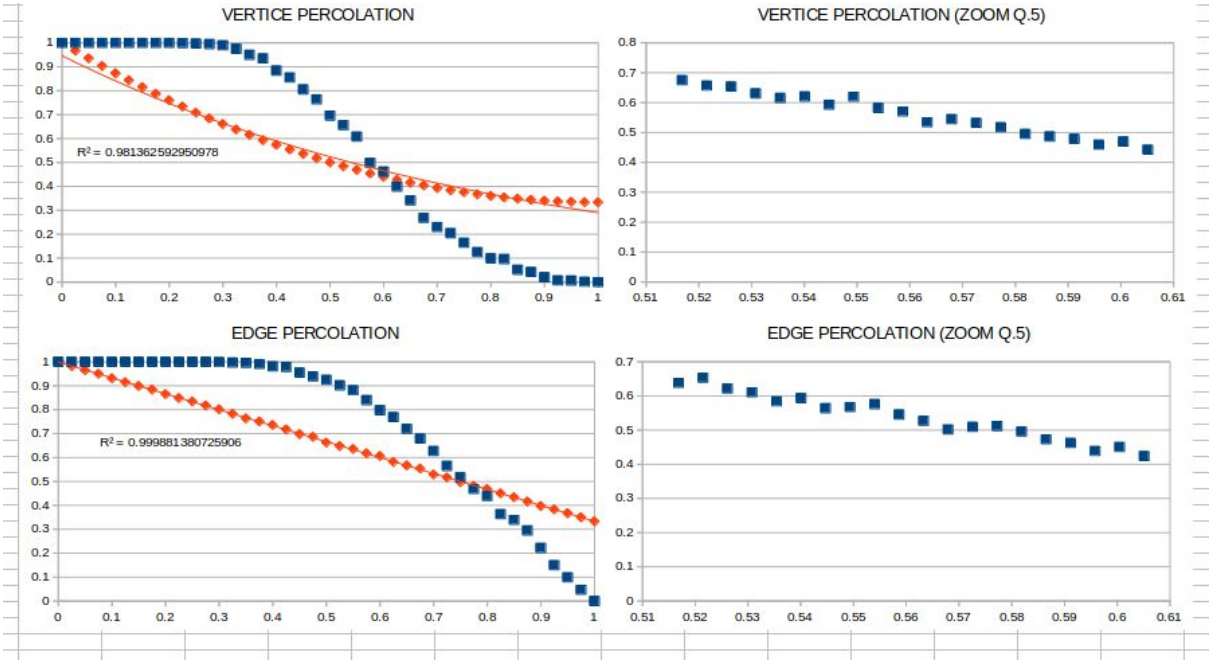
Per altra banda, el pendent de la línia blava al creuar $Y = 0,5$ és cada cop menor com major és el nombre de vèrtex del graf.

Els valors de Q.5 en la percolació per vèrtex s'acosten a un valor proper a 0.9 i en la percolació per arestes s'acosten a un valor proper a 1. Per aquest motiu aquest experiment s'assembla molt al de la graella descrit anteriorment.

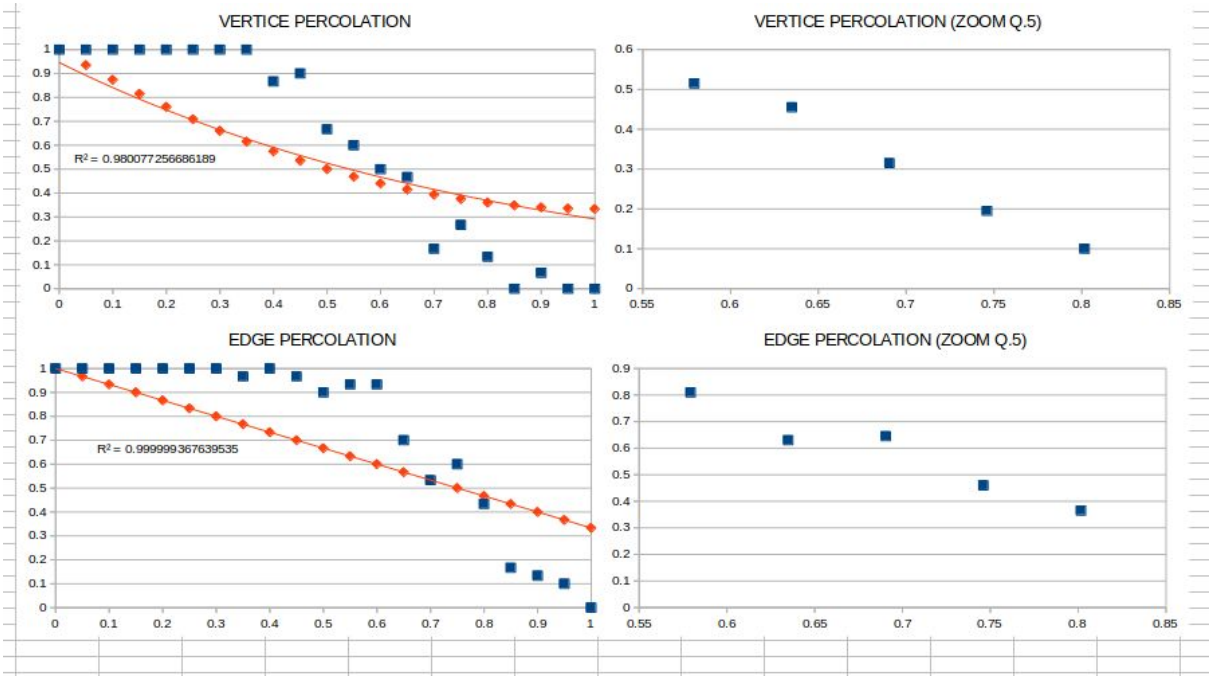
`nombre de vèrtexs => 10`




`nombre de vèrtexs => 1000`



`nombre de vèrtexs => 1000000`




`conclusions`

En aquest experiment el comportament de la transició de fase ha variat notablement respecte als dos casos de travessia analitzats anteriorment ja que el pendent de la corba en la transició de fase sembla no estar relacionat amb *N*.

Per altra banda en els casos de travessia anteriors, s'aproximaven a un valor Q.5 constant quan N tendia a infinit però ho feien per la l'esquerra. Observant el gràfic de zoom podem veure que per la percolació de vèrtex ens aproximem per la dreta:

$$N = 10,\ Q.5 \approx 0.855 \mid N = 1000,\ Q.5 \approx 0.58 \mid N = 100.000\ ,\ Q.5 \approx 0.555$$

I també en el cas de la percolació per arestes:

$$N = 10,\ Q.5 \approx 0.865 \mid N = 1000,\ Q.5 \approx 0.58 \mid N = 100.000\ ,\ Q.5 \approx 0.655$$

## Cicles + Barabasi-Albert

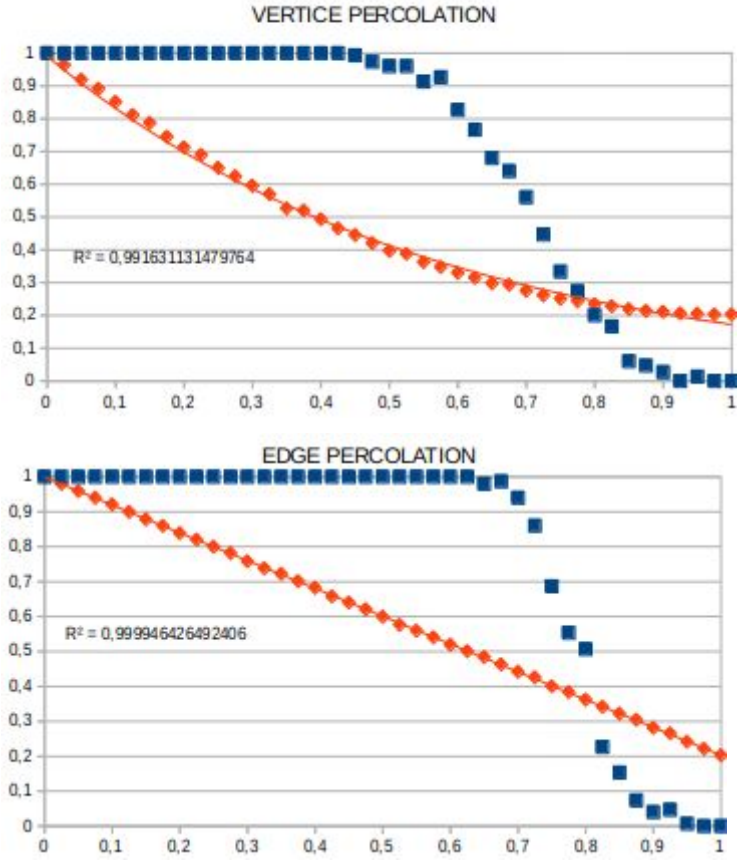
Per a la generació en python d'un graf Barabasi-Albert necessitem els paràmetres N i M, per a aquest experiment hem escollit una $M = 3$ constant. D'aquesta manera l'única variable que varia és la N (nombre de vèrtexs).

Les línies blaves representen el compliment de la propietat del graf i les vermelles representen la suma de (V + E) respecte al valor màxim de vèrtexs i arestes que pot tenir el graf (per exemple en el cas on q = 0 no hem eliminat cap aresta de manera que aquest coeficient serà 1).

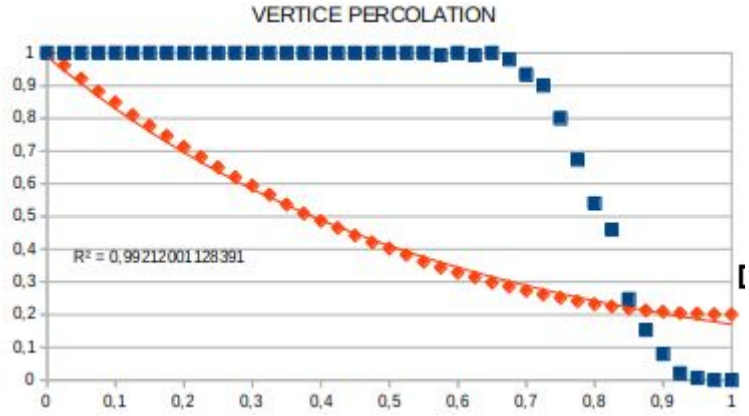
L'eix X representa la probabilitat de fallida (q) i l'eix Y representa el tant per u.

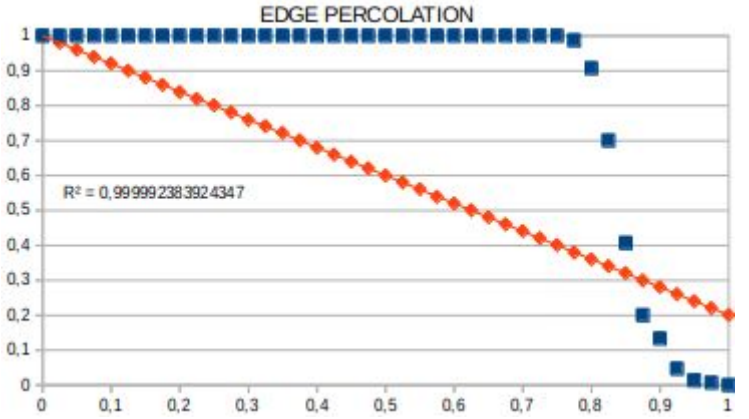
En aquest cas no hem fet servir zoom ja que creiem que en els experiments anteriors ja s'ha vist amb prou claredat la seva funció.

```
nombre de vèrtexs => 128
```

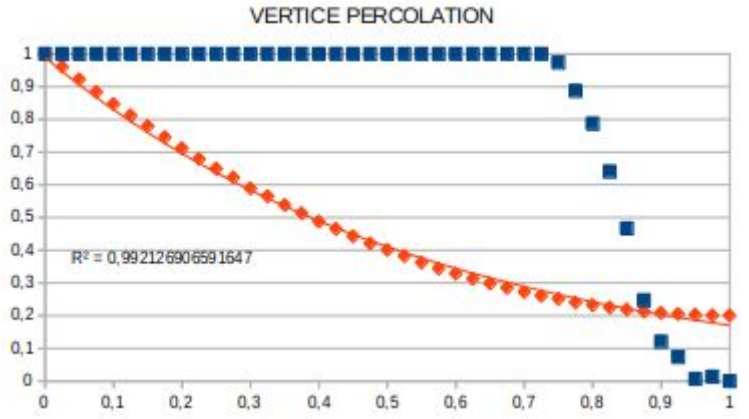


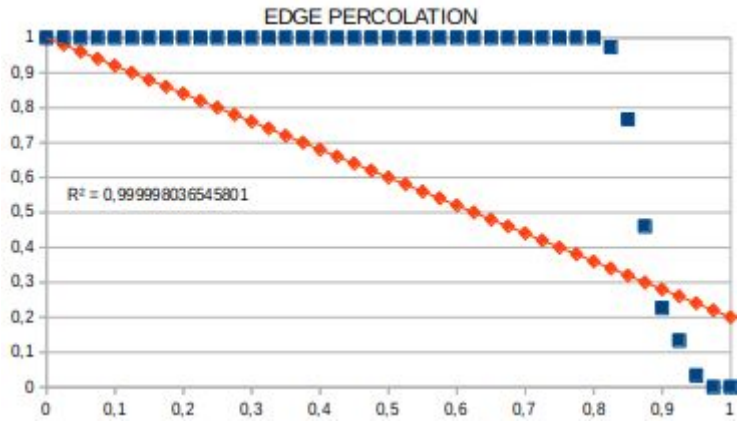
nombre de vèrtexs => 1024




nombre de vèrtexs => 4096






`conclusions`

En aquestes gràfiques podem veure fàcilment com es produeix una transició de fase amb *q* relativament altes. Podem deduir que el graf Barabasi-Albert inicial conté una quantitat bastant alta de cicles, fet que fa que encara que en el procés de percolació es suprimeixi un vèrtex o aresta pertanyent a un cicle el graf segueixi tenint cicles.

També podem observar que en la percolació per vèrtexs la transició de fase dura més, sobretot amb *N* baixes.

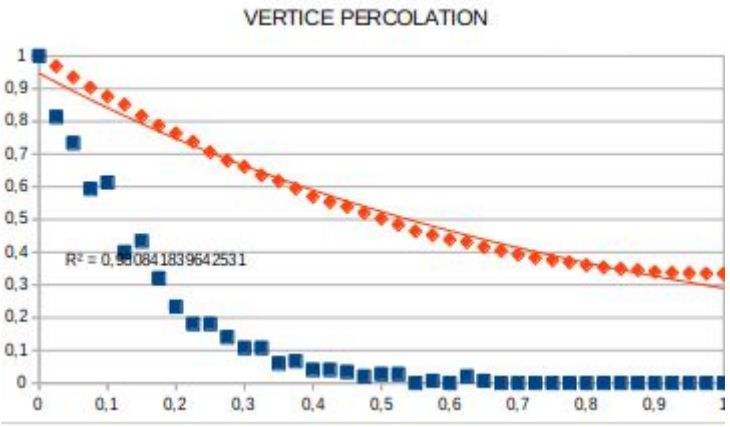
## Cicles + Watts–Strogatz

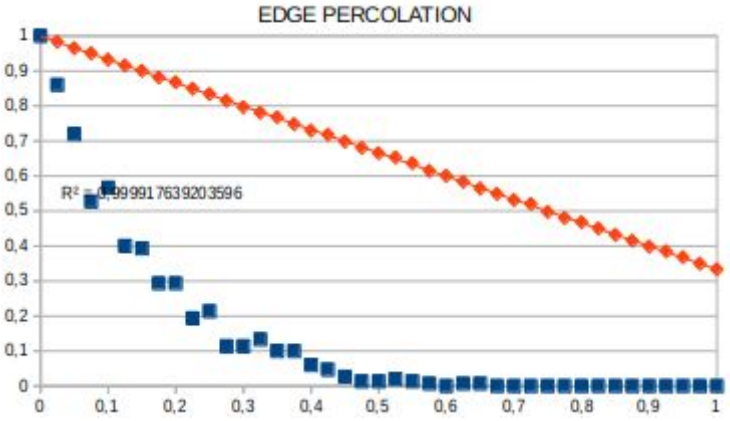
Com hem explicat anteriorment el generador de grafs en python ens obliga a establir una *K* i una *P*, per aquest experiment hem fet servir els valors 3 i 0,5 respectivament. D'aquesta manera l'única variable que varia és la N (nombre de vèrtexs).

Les línies blaves representen el compliment de la propietat del graf i les vermelles representen la suma de (V + E) respecte al valor màxim de vèrtexs i arestes que pot tenir el graf (per exemple en el cas on q = 0 no hem eliminat cap aresta de manera que aquest coeficient serà 1).
L'eix X representa la probabilitat de fallida (q) i l'eix Y representa el tant per u.

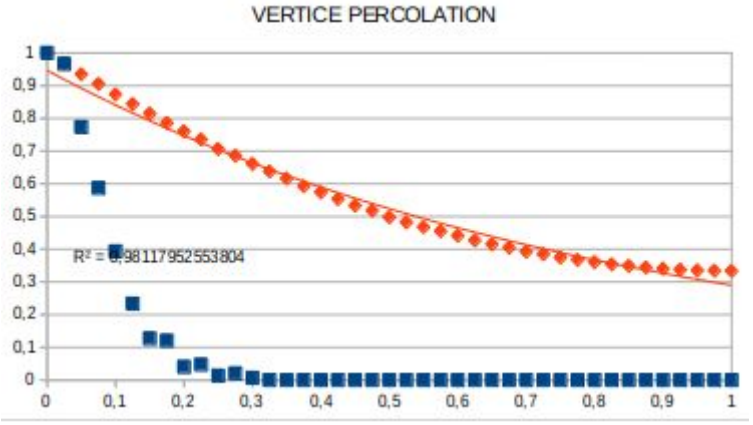
En aquest cas no hem fet servir zoom ja que creiem que en els experiments anteriors ja s'ha vist amb prou claredat la seva funció.

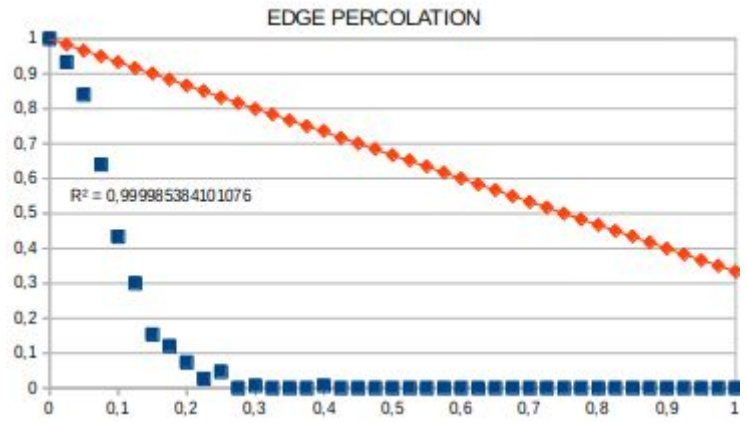
nombre de vèrtexs => 128



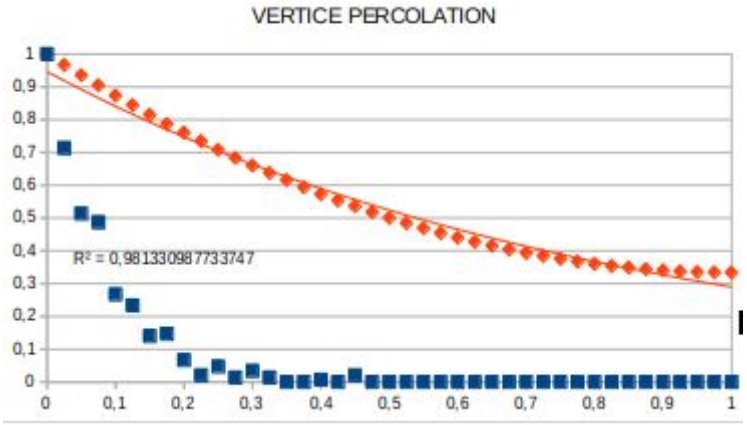



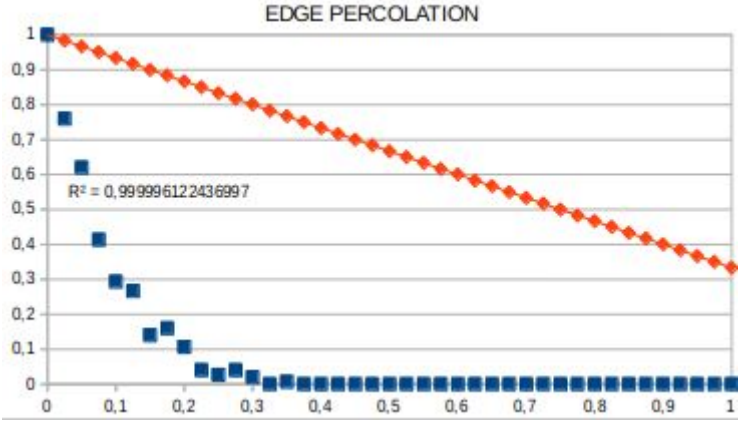

nombre de vèrtexs => 1024






nombre de vèrtexs => 4096



## conclusions

En aquestes gràfiques podem veure que per als grafs aleatoris Watts-Strogatz és difícil afirmar que existeix una transició de fase per a la propietat dels cicles. Més aviat podem veure com el compliment de la propietat disminueix exponencialment, encara que per a una $N$ igual a 1024 sí que sembla que comença amb un petit període en el qual es compleix la propietat.

Creiem que el motiu principal pel qual passa això és que els grafs generats tenen molt pocs cicles, per tant és possible que si el graf inicial tingués més cicles podríem observar la transició de fase.

# Conclusions i dificultats

En la realització d'aquest estudi hem hagut de superar diferents reptes inicials, com ara decidir la implementació de les estructures de dades, escriure uns algorismes eficients i escollir els tipus de models de grafs a estudiar.

Per altra banda errors com per exemple l'ús de variables double a l'hora de calcular probabilitats ens han endarrerit i alhora ens han obligat a millorar el codi utilitzant fraccions..

Una de les principals dificultats amb les quals ens hem trobat durant la realització d'aquest treball ha estat el fet d'una implementació inicial de la classe graf no del tot modular de cara a l'estudi posterior de diferents propietats que, degut a la falta de temps, ens ha impedit realitzar certs experiments amb altres propietats com per exemple, estudiar la transició de fase per a grafs aleatoris sobre la propietat de si són o no connexos.

Durant l'execució dels programes ens vam adonar que per a grafs amb molts vèrtexs i arestes el temps d'execució del programa era massa elevat degut, principalment, a la gran quantitat d'experiments que realitzàvem per cada graf. És per això que vam decidir paral·lelitzar el programa però, per manca de temps i recursos, no vam poder realitzar-ho amb els resultats esperats.

Un cop superats aquests obstacles hem hagut de fer un tractament adequat de les dades obtingudes. El problema que teníem era que en l'estudi de la transició de fase es perdia molt de temps en obtenir dades que no eren del tot rellevants. La solució que hem escollit ha estat fer un *zoom* en la zona en què les dades són més rellevants per als nostres estudis. Això ens ha permès obtenir les dades amb més facilitat i sense perdre precisió.

En definitiva aquest treball ens ha servit per a millorar el nostre coneixement sobre les estructures de dades més adients que podem usar per al processament eficient de grafs i entendre com es duen a terme els estudis de grafs mitjançant l'obtenció massiva de dades per al seu posterior estudi.

# Referències


[1]"Grid Graph -- from Wolfram MathWorld", *Mathworld.wolfram.com*. [Online]. Available: http://mathworld.wolfram.com/GridGraph.html.

[2]"Random graph", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Random_graph.

[3]R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Addison-Wesley, 2011.

[4]"Reference — NetworkX 2.3 documentation", *Networkx.github.io*. [Online]. Available: https://networkx.github.io/documentation/stable/reference/index.html.

[5]"Barabasi Albert Graph (for Scale Free Models) - GeeksforGeeks", *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/barabasi-albert-graph-scale-free-models/.

[6]"Scale-free network", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Scale-free_network.

[7]"Watts–Strogatz model", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Watts%E2%80%93Strogatz_model.

[8]"Transición de fase", *Es.wikipedia.org*. [Online]. Available: https://es.wikipedia.org/wiki/Transici%C3%B3n_de_fase.

[9]"Teoría de grafos", *Es.wikipedia.org*. [Online]. Available: https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos.

[10]"Small-world network", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Small-world_network.