Algorismia

Estudi Experimental de Connectivitat i Percolació de Grafs

Pau Belda, Guillem Cabré, Marc Peñalver, Prisca Oleart

Curs 2024-25, Quatrimestre de tardor

Continguts

1 Definicions

1.1 Percolació

La percolació en un graf G consisteix en eliminar o desactivar nodes o arestes, i posteriorment es mesura com això afecta una certa propietat global del graf. Quan desactivem una aresta o un node, direm que ha tingut una fallida.

En termes generals, l'objectiu és estudiar com el graf passa d'estar completament connectat a parcialment o totalment desconnectat a mesura que es treuen alguns dels seus components. Quan parlem de percolació, considerem una probabilitat p que determina si una component del graf (node o aresta) es desactiva aleatòriament. Aquest mecanisme és especialment rellevant per a l'estudi de xarxes complexes, ja que ens ajuda a comprendre com de robust o vulnerable és el sistema que volem analitzar.

- **Percolació per nodes**: Cada node té una probabilitat p de ser desactivat. Un cop fet això, s'analitza com ha canviat la connectivitat del graf.
- **Percolació per arestes**: En aquest cas, les arestes es desactiven en lloc dels nodes. Això també afecta la connectivitat, ja que les connexions directes entre nodes es perden.

Després d'aplicar el procés de percolació (sobre nodes o arestes), s'obté un graf percolat, que és la versió modificada del graf original, amb una connectivitat reduïda i, possiblement, components desconnectats. Aquest graf l'anomenarem G_{p}.

1.2 Transició de Fase

Una transició de fase d'un graf per a una propietat concreta Π fa referència a un resultat satisfactori d'un procés de percolació aplicat al graf. En el nostre cas aquesta propietat Π serà la connectivitat del graf.

Definim un resultat com a satisfactori si, donat que es troba una probabilitat de valor q tal que es compleix la propietat Π al graf G_q (definim aquesta probabilitat com q_Π), per als grafs $G_{q'}$ on $q' > q_\Pi$, aquests verifiquen la propietat Π, i als grafs $G_{q'}$ on $q' < q_\Pi$, no la verifiquen (ambdues afirmacions són vàlides si es compleixen amb una probabilitat prou alta).

Quan s'ha obtingut aquest resultat, diem que la propietat Π presenta una transició de fase al voltant de q_Π.

1.3 Objectius de la Experimentació

En aquesta secció s'exposen els grafs seleccionats per a l'estudi experimental. Es detallaran les característiques específiques de cada graf, així com els algorismes utilitzats per a la seva generació. Posteriorment, explicarem per què aquests grafs han estat considerats els més rellevants per al nostre estudi.

2 Grafs Seleccionats

En aquesta secció s'exposen els grafs seleccionats per a l'estudi experimental. Concretament, explicarem les peculiaritats de cada graf. Així mateix, es detallaran els algorismes utilitzats per a la generació de cada tipus de graf. Posteriorment, exposarem per que aquests son els que mes ens han semblat interresants per fer l'estudi.

2.1 Erdős-Rényi

Algorisme 1 Generació de graf Erdős-Rényi $G(n, p)$

Entrada: n (dimensió de la graella), p (probabilitat d'establir aresta entre nodes)
Sortida: Graf g

```
1: Inicialitzar el graf g amb n nodes
2: for i = 0 fins a n − 1 do
3:     for j = i + 1 fins a n − 1 do
4:         if rand01() < p then
5:             Afegir una aresta entre i i j al graf g
6:         end if
7:     end for
8: end for
9: Retornar el graf g
```

El generador de grafs Erdős-Rényi crea un graf aleatoriitzant la connexió entre nodes. Sigui p la probabilitat d'establir una aresta entre qualsevol parell de nodes del graf. Per a cada parell de nodes (i, j), es genera un nombre aleatori mitjançant la funció $rand01()$, que retorna un valor entre 0 i 1. Si aquest valor és menor que p, s'estableix una aresta entre i i j. Aquest procés es repeteix per a tots els parells possibles de nodes.

La instrucció $j = i + 1$ en el bucle interior assegura que només es considerin les arestes entre nodes diferents i evita la duplicació d'arestes. Això és important perquè en un graf no dirigit, l'aresta entre i i j és la mateixa que l'aresta entre j i i. D'aquesta manera, es redueix el nombre de connexions a calcular i s'assegura que cada aresta es consideri només una vegada.

2.2 Graella Quadrada

Algorisme 2 Generació de Graf de Graella Quadrada $G(m \times m)$

Entrada: m (dimensió de la graella)
Sortida: Graf g

Inicialitzar $n = m \times m$ (nombre de nodes del graf)
Crear un graf buit g amb n nodes
for $i = 0$ fins a $m - 1$ **do**
    for $j = 0$ fins a $m - 1$ **do**
        $nodeActual = i \times m + j$
        if $i < m - 1$ **then**
            $nodeFilaInferior = (i + 1) \times m + j$
            Afegir una aresta entre $nodeActual$ i $nodeFilaInferior$
        end if
        if $j < m - 1$ **then**
            $nodeColumnaDreta = i \times m + (j + 1)$
            Afegir una aresta entre $nodeActual$ i $nodeColumnaDreta$
        end if
    end for
end for
Retornar el graf g

El generador de grafs de graella quadrada crea un graf amb una estructura regular en què cada node és adjacent als nodes de la fila superior, inferior, esquerra i dreta, si aquests existeixen. Això s'aconsegueix comprovant per cada node si existeixen nodes a la fila inferior i a la columna dreta, i si és així, s'afegeixen les arestes corresponents.

2.3 Graella Triangular

Algorisme 3 Generació de Graf de Graella Triangular $G(rows)$

Entrada: $rows$ (nombre de files)
Sortida: Graf g

Inicialitzar $n = \frac{rows \times (rows+1)}{2}$ (nombre de nodes del graf)
Crear un graf buit g amb n nodes
$nodeActual = 0$
for $i = 0$ fins a $rows - 1$ **do**
    for $j = 0$ fins a i **do**
        if $j < i$ **then**
            $nodeDreta = nodeActual + 1$
            Afegir una aresta entre $nodeActual$ i $nodeDreta$
        end if
        if $i < rows - 1$ **then**
            $nodeInferiorEsquerra = nodeActual + i + 1$
            Afegir una aresta entre $nodeActual$ i $nodeInferiorEsquerra$
            $nodeInferiorDret = nodeActual + i + 2$
            Afegir una aresta entre $nodeActual$ i $nodeInferiorDret$
        end if
        $nodeActual = nodeActual + 1$
    end for
end for
Retornar el graf g

El generador de grafs de graella triangular crea un graf amb n nodes, on $n = 1 + 2 + 3 + \ldots + rows = \frac{rows \times (rows+1)}{2}$. Aquests nodes tenen una estructura en què cada node està connectat als veïns de la dreta i l'esquerra, així com als nodes superiors i inferiors, tant a l'esquerra com a la dreta, sempre que aquests existeixin. Això s'aconsegueix comprovant per a cada node si hi ha nodes a la fila inferior i a la columna dreta, i si és així, s'afegeixen les arestes corresponents.

2.4 Graf Geomètric Aleatori

Algorisme 4 Generació de Graf Geomètric Aleatori $G(n, r)$

Entrada: n (nombre de nodes), r (radi de connexió)
Sortida: Graf g

1: Crear un graf buit g amb n nodes
2: Inicialitzar un vector de coordenades *coords* de longitud n
3: **for** $i = 0$ fins a $n - 1$ **do**
4: $\quad coords[i].x = \text{rand01}()$
5: $\quad coords[i].y = \text{rand01}()$
6: **end for**
7: **for** $i = 0$ fins a $n - 1$ **do**
8: $\quad$ **for** $j = i + 1$ fins a $n - 1$ **do**
9: $\quad\quad dist_x = coords[i].x - coords[j].x$
10: $\quad\quad dist_y = coords[i].y - coords[j].y$
11: $\quad\quad$ **if** $\sqrt{dist_x^2 + dist_y^2} < r$ **then**
12: $\quad\quad\quad$ Afegir una aresta entre i i j al graf g
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end for**
16: Retornar el graf g

El generador de grafs geomètrics aleatoris crea un graf amb n nodes, on cada node es col·loca aleatòriament en un espai bidimensional unitari. Dos nodes estan connectats per una aresta si la distància entre ells és menor que un radi r especificat. Això es determina calculant la distància euclidiana entre tots els parells de nodes i afegint arestes quan la distància és inferior a r.

2.5 Graf de Barabási-Albert

Algorisme 5 Generació de Graf de Barabási-Albert $G(n, m_0, m)$

Entrada: n (nombre de nodes), m_0 (nombre de nodes inicials), m (grau de connexió per nou node)
Sortida: Graf g

Crear un graf buit g amb n nodes
Inicialitzar un vector *connection_degree* de longitud n
for $i = 0$ fins a $m_0 - 1$ **do**
    for $j = i + 1$ fins a $m_0 - 1$ **do**
        Afegir una aresta entre i i j al graf g
    end for
    $connection\_degree[i] = m_0 - 1$
end for
for $i = m_0$ fins a $n - 1$ **do**
    Inicialitzar un vector buit *candidates*
    while la longitud de *candidates* és menor que m **do**
        $selectedNode$ = preferentialAttachment($connection\_degree$)
        if $selectedNode$ no està en *candidates* **then**
            Afegir $selectedNode$ a *candidates*
        end if
    end while
    for cada j en *candidates* **do**
        Afegir una aresta entre i i j al graf g
        $connection\_degree[j] += 1$
    end for
    $connection\_degree[i] = m$
end for
Retornar el graf g

El generador de grafs de Barabási-Albert crea un graf que segueix el model de creixement de xarxes, on s'afegeixen nodes nous que es connecten a nodes existents en funció del seu grau de connexió. Comença amb un conjunt inicial de m_0 nodes completament connectats, i cada nou node que s'afegeix selecciona m nodes existents per connectar-se, amb una probabilitat proporcional al seu grau de connexió.

Aquesta selecció es realitza mitjançant la funció `preferentialAttachment`, que s'encarrega de seleccionar un node existent basant-se en el seu grau de connexió. La funció funciona de la següent manera:

Algorisme 6 Preferential Attachment

Entrada: *connection_degree* (vector de graus de connexió)
Sortida: *chosen* (node seleccionat)
Inicialitzar $degree\_sum = 0$, $temp\_sum = 0$
for cada i en *connection_degree* **do**
    $degree\_sum \mathrel{+}= i$
end for
$random\_num$ = rand() mod $degree\_sum$
for $i = 0$ fins a length(connection_degree) - 1 **do**
    $temp\_sum \mathrel{+}= connection\_degree[i]$
    if $random\_num \leq temp\_sum$ **then**
        $chosen \leftarrow i$

    end if
end for
Retornar *chosen*

Aquesta funció calcula la suma total dels graus de connexió de tots els nodes existents i selecciona

aleatòriament un node, on la probabilitat de seleccionar cada node és proporcional al seu grau de connexió. Així, els nodes amb més connexions tenen una major probabilitat de ser seleccionats, promovent el creixement de xarxes amb característiques d'escala.

3 Algoritmes

En aquesta secció es presenten els principals algorismes emprats en l'estudi de la transició de fase. A continuació, detallarem cada algorisme i donarem una breu explicació d'aquests. Els algorismes abordats són els següents:

3.1 Percolació per Arestes

Algorisme 7 Percolació d'Arestes en un Graf

Entrada: Graf $G = (V, E)$ amb n nodes, probabilitat p
Sortida: Graf percolat $G' = (V, E')$ on $E' \subseteq E$

1: Crear un nou graf buit G' amb n nodes (còpia profunda de G)
2: **for** cada node u en V **do**
3: **for** cada node v en la llista d'adjacència d'u d'el graf G **do**
4: **if** $u < v \ \wedge \ rand01() > p$ **then**
5: Afegir l'aresta (u, v) al graf G'
6: **end if**
7: **end for**
8: **end for**
9: Retornar el graf G'

Aquest algorisme retorna el graf percolat a partir del graf original. Això s'aconsegueix eliminant les arestes amb una probabilitat p. Per evitar processar una mateixa aresta més d'una vegada, s'utilitza la condició $u < v$, ja que en grafs no dirigits una aresta (u, v) és equivalent a (v, u). Tal com s'ha mencionat prèviament, la funció `rand01()` genera un nombre aleatori entre 0 i 1, que es compara amb la probabilitat p per decidir si es manté o s'elimina una aresta.

3.2 Percolació per Nodes

Algorisme 8 Percolació de Nodes en un Graf

Entrada: Graf $G = (V, E)$ amb n nodes, probabilitat p
Sortida: Graf percolat $G' = (V', E')$ on $V' \subseteq V$ i $E' \subseteq E$

Inicialitzar un vector $posicioNodes$ de longitud n
Assignar $nbNodesVius = n$
for cada node u en V **do**
    Generar un valor aleatori $r =$ `rand01()`
    if $r < p$ **then**
        Marcar el node u com a fallat
        Decrementar $nbNodesVius$
    else
        Actualitzar la posició del node viu $posicioNodes[u] = u - n + nbNodesVius$
    end if
end for

Crear un graf buit G' amb $nbNodesVius$ nodes
for cada node u en V **do**
    if u no ha fallat **then**
        for cada node v en la llista d'adjacència de u en G **do**
            if v no ha fallat $\wedge$ $u < v$ **then**
                Afegir l'aresta $(posicioNodes[u], posicioNodes[v])$ a G'
            end if
        end for
    end if
end for
Retornar el graf G'

Aquest algorisme retorna el graf percolat a partir del graf original eliminant nodes amb una probabilitat p. Per a cada node, es genera un valor aleatori entre 0 i 1 mitjançant la funció `rand01()`. Si aquest valor és inferior a p, el node es considera eliminat (fallat) i no es conservarà en el graf resultant.
Els nodes que sobreviuen són reindexats per assegurar que el nou graf té una numeració consecutiva de nodes. Les arestes només es conserven si ambdós nodes que connecten han sobreviscut, i es manté la condició $u < v$ per evitar afegir la mateixa aresta dues vegades, ja que en els grafs no dirigits una aresta (u, v) és equivalent a (v, u). Així, el resultat és un graf amb una mida reduïda en funció de la probabilitat p, mantenint només els nodes i arestes que han "sobreviscut" al procés de percolació.

3.3 Càlcul de Components Connexes

Algorisme 9 Càlcul del Nombre de Components Connexes

Entrada: Graf $G = (V, E)$ amb n nodes
Sortida: Nombre de components connexos $componentCount$

Inicialitzar un vector $visited$ de longitud n amb valors $false$
Inicialitzar $componentCount = 0$
for cada node $i = 0$ fins a $n - 1$ **do**
    if el node i no ha estat visitat **then**
        Incrementar $componentCount$
        Realitzar una DFS a partir del node i, marcant els nodes visitats
    end if
end for
Retornar $componentCount$

L'algorisme utilitza una cerca en profunditat (DFS) per explorar cada component connex del graf. Cada vegada que es troba un node no visitat, es crida la funció `dfs` per explorar recursivament tots els nodes connectats a aquest node. Aquesta crida recursiva assegura que tots els nodes del mateix component quedin marcats com a visitats, evitant comptar-los més d'una vegada.

4 Experimentació

Per dur a terme l'experimentació del projecte, hem utilitzat diferents eines. Hem programat dos programes en `C++`, un llenguatge que ens ofereix molta eficàcia temporal i espacial. Aquests programes són el `main` i el `runner`. També hem dissenyat un fitxer de classe `graph` amb tots els atributs i funcions necessàries per operar amb els grafs. Aquesta classe representa els grafs com a llistes d'adjacència.

Per compilar aquests programes, hem fet ús del programari lliure `make`, que automatitza i paral·litza el compilatge i l'enllaç.

A més, hem dissenyat scripts per a l'interpret `R`, que és un programari de tractament de dades que ens analitzarà i generarà gràfics dels resultats dels estudis, que estaran en format `.csv`.

Més informació del procés d'experimentació es pot trobar en el GitHub del projecte, premeu `aquí` per accedir-hi. Allà, a part del codi, també podreu consultar més informació sobre la generació de grafs, les dependències del programa per compilar-lo i executar-lo, com inserir els paràmetres pels programes i més.

4.1 Metodologia

El programa `main`, mitjançant la classe `graph`, ens ha permès analitzar les propietats del canvi de fase a partir dels paràmetres inicials. Aquests paràmetres són els següents:

- **RandomSeed**: La llavor per al generador aleatori.
- **NúmeroMínimNodes**: El nombre mínim de nodes del graf.
- **NúmeroMàximNodes**: El nombre màxim de nodes del graf.
- **NúmeroNodesStep**: Increment dels nodes en cada iteració.
- **IteracionsPerObtenirResultat**: El nombre de vegades que es provarà la configuració per probabilitat p de percolació i per nombre de vèrtex n.
- **ModePercolació**: Tipus de percolació per nodes o per arestes.
- **PathResultat**: Fitxer on es guardaran els resultats.
- **AlgorismeGeneradorGraf**: Algoritme utilitzat per generar el graf (per exemple, Erdős-Rényi, Square-Grid, etc.).
- **ParàmetresAlgorisme**: Paràmetres addicionals per al generador de graf (opcional segons l'algorisme).

A partir d'aquests paràmetres, el programa `main` escriurà un fitxer `PATH.csv` que posteriorment serà analitzat mitjançant el software de tractament de dades `R`.

Per altra banda, tenim el programa `runner`, que rebrà com a input un fitxer de text. Aquest fitxer tindrà un llistat de paràmetres per diferents experiments del programa `main`. Un exemple d'això seria:

```
RGN   MIN MAX  STEP ITs   PERC-MODE  RESULT-PATH      GEN-ALGORITM    PARAMETERS-GEN
-------------------------------------------------------------------------------------
21312 10  100   10  1000  NODE_PERC  ./data/test1.csv Erdos-Renyi        0.1
35353 50  500   50  1000  EDGE_PERC  ./data/test2.csv Random-Geometric   0.3
72479 100 1000  100 100   EDGE_PERC  ./data/test3.csv Square-Grid
```

El programa `runner`, per cada fila del fitxer que rep, inicialitzarà una instància del programa `main`, aconseguint d'aquesta manera automatitzar molt més els tests, podent córrer diferents programes `main` simultàniament.

4.2 Programa Main

Per entendre els resultats també s'ha d'entendre les decisions que s'han pres per la recollida de dades. Analitzarem el programa `main` mitjançant un pseudocodi per no entrar en conceptes avançats de `C++`. A continuació vegeu una mostra del pseudocodi:

Algorisme 10 Descripció de l'experiment

```
1: Seleccionar opcions de configuració
2: Inicialitzar el generador de nombres aleatoris
3: Obrir l'arxiu CSV i escriure la capçalera
4: for n in range(MIN_NB_NODES, MAX_NB_NODES + 1, NB_NODES_STEP) do
5:     for p des de 0 fins a 1 amb pas 0.01 do
6:         Inicialitzar el comptador de grafs connexos
7:         for i = 0 fins a TRIES_PER_P do
8:             repeat
9:                 Generar el graf seleccionat(p, n, params)
10:            until el graf és connex
11:            Aplicar percolació (per nodes o arestes) al graf
12:            if el graf percolat és connex then
13:                Incrementar el comptador de grafs connexos
14:            end if
15:        end for
16:        Escriure entrada resultant al CSV
17:    end for
18: end for
```

Ara, analitzarem el codi. Per començar, el programa preguntarà per totes les opcions necessàries. D'aquesta manera, ens podem permetre tenir un sol programa que pugui fer tot el que necessitem i que sigui altament modular. S'utilitzarà la llavor per generar nombres aleatoris, i així l'experiment podrà ser repetit amb els mateixos resultats. Després, crearà el fitxer `PATH.csv`, al qual s'hi inseriran entrades que posteriorment s'analitzaran.

Ara analitzarem l'algorisme encarregat de generar els resultats. Vegeu com primerament iterarem sobre n tantes vegades com s'hagi indicat en la entrada. Alhora, també iterarem per cada n sobre una probabilitat de fallida de percolació. Aquest bucle tindrà 100 iteracions, $p \in \{0.00, 0.01, \ldots, 1.00\}$. A més d'aquests dos bucles, iterarem una altra vegada sobre p i n tantes vegades com l'usuari hagi indicat en l'apartat *IteracionsPerObtenirResultat*. Així, es farà una mitjana amb més o menys mostres.

S'iniciarà un comptador a 0 que representarà el nombre de grafs percolats connexos. S'utilitzarà el generador de grafs seleccionat a les opcions per generar el graf que posteriorment serà percolat. Vegeu que aquí generarem grafs fins a aconseguir un graf connex. Aquesta decisió la vam prendre per tenir una representació més acurada de la transició de fase. Més endavant, es tornarà a considerar aquesta decisió, ja que hi ha grafs, com ara el *Random Geometric Graph*, que requereixen un paràmetre r, el qual, amb valors petits de r, acostuma a generar grafs no connexos.

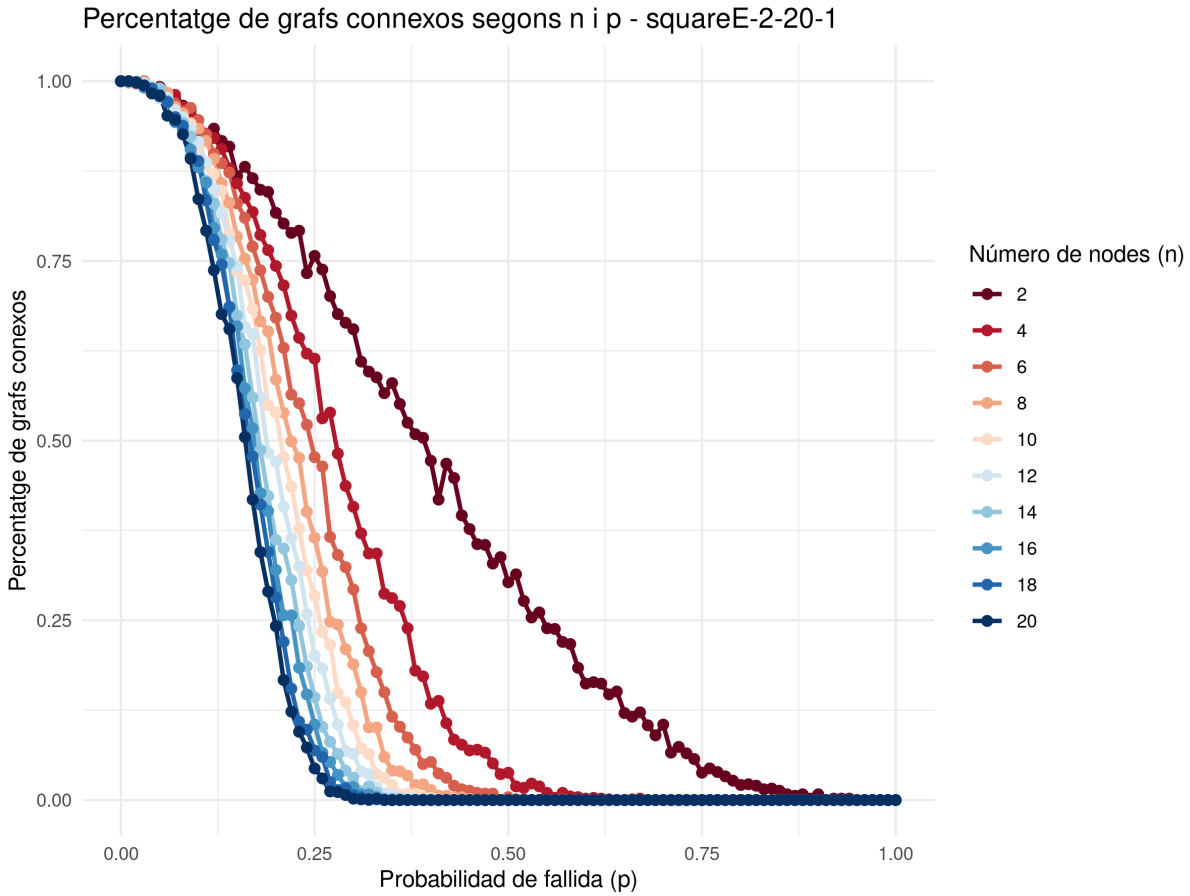
Per acabar, percolarem el graf $G(V, E)$ de la manera que s'hagi especificat a l'input, ja sigui per nodes o per arestes, obtenint G_{p}. A G_{p} se li aplicarà un algorisme que determinarà si el graf és connex. Si ho és, s'incrementarà el comptador. Quan les *IteracionsPerObtenirResultat* s'hagin completat, s'escriurà l'entrada resultant al fitxer `PATH.csv`.

4.3 Graella quadrada

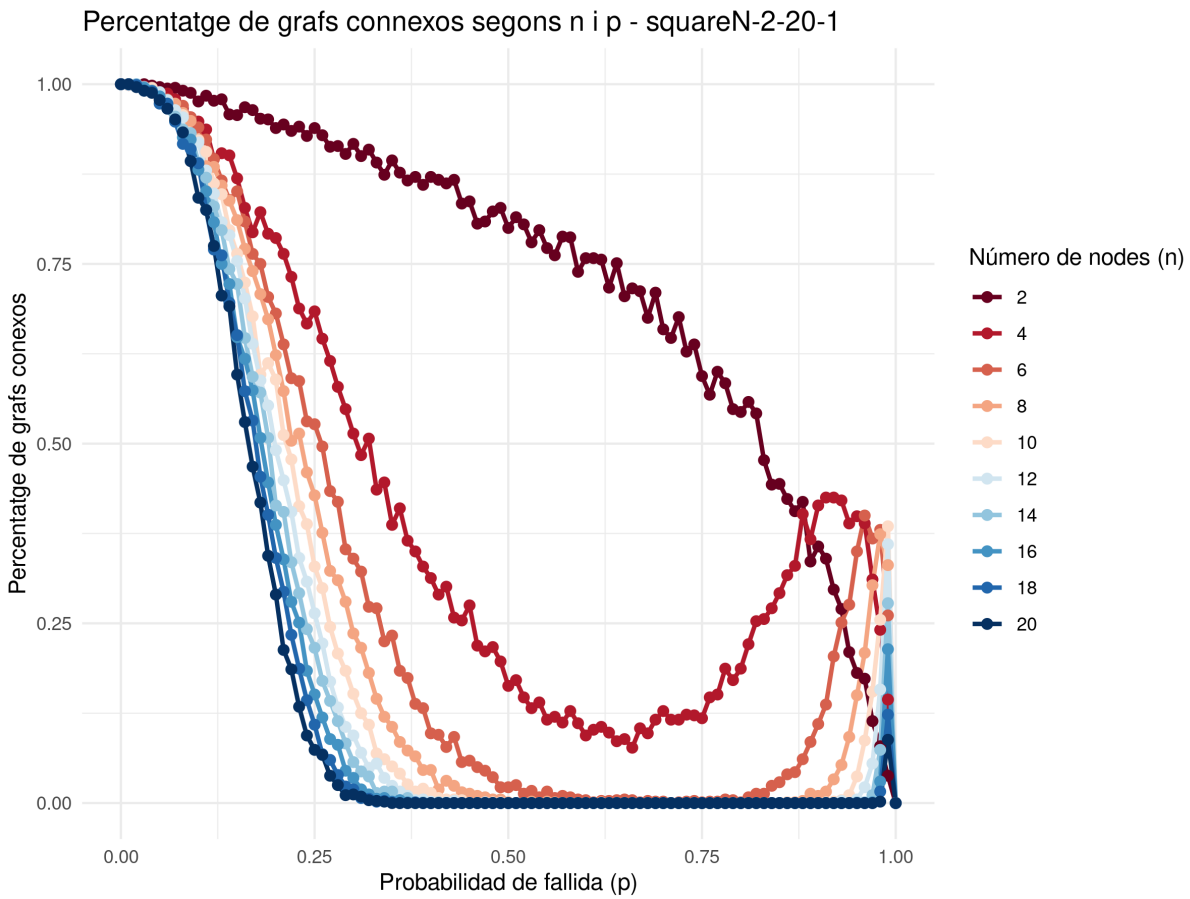
Per estudiar la possible transició de fase a graelles quadrades hem decidit fer proves amb grafs de mida petita, des de 2x2 nodes a 20x20, així com amb grafs de mida més gran, de 20x20 nodes a 200x200. Això ens servirà per estudiar el comportament de la percolació tant per node com per aresta en diferents escales de complexitat. Hem triat aquestes mides de grafs per poder comparar resultats en diferents dimensions i observar com la mida del graf influeix en la percolació, formació de components connexes i transició de fase.

4.3.1 Mida petita

Percolació per aresta

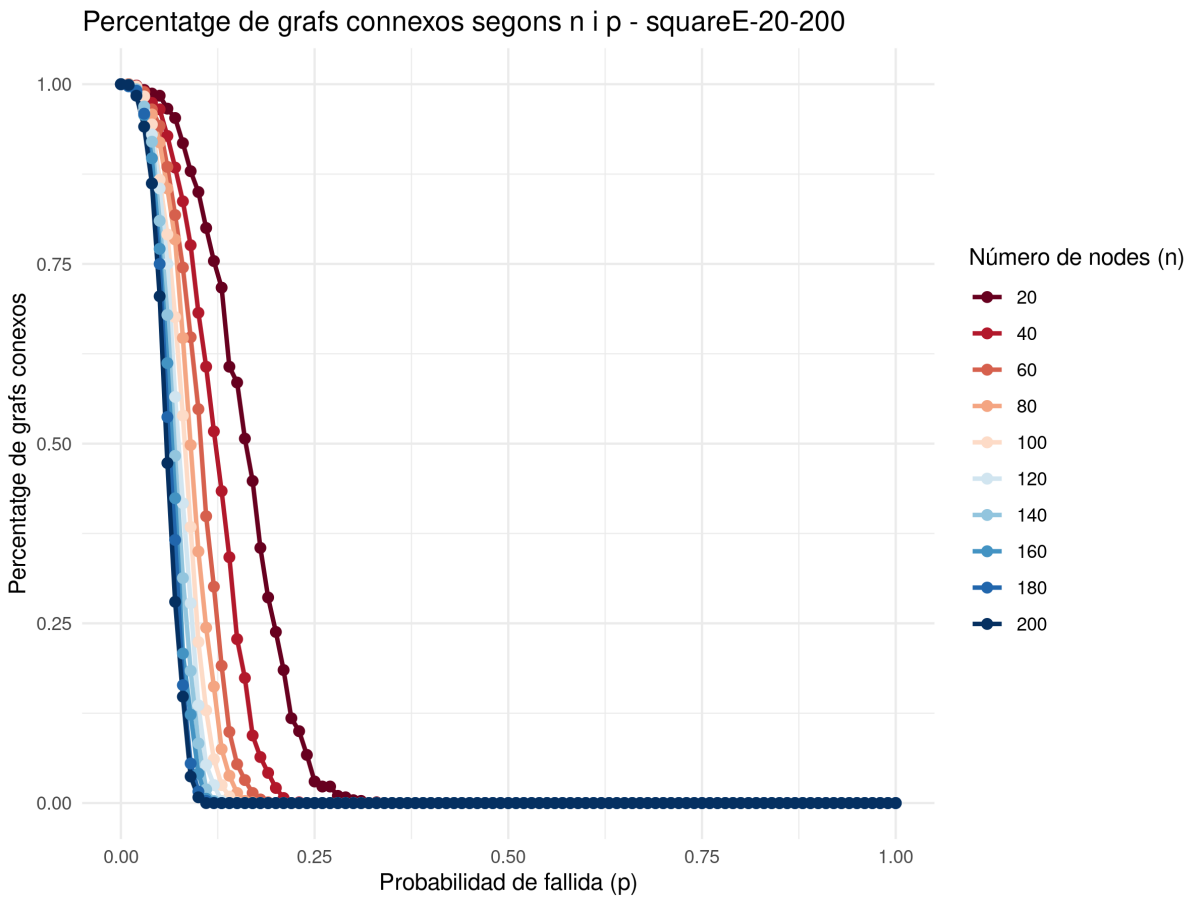


Percolació per node

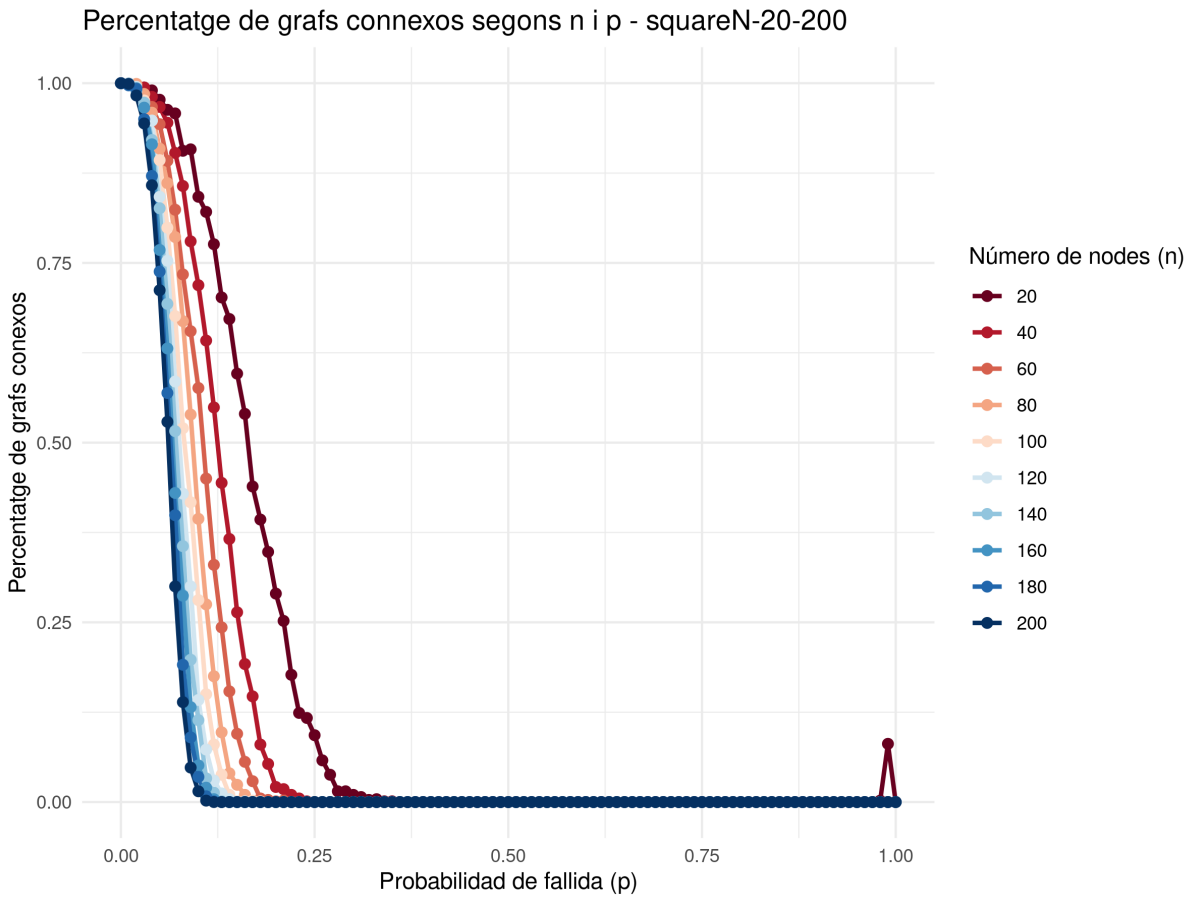


4.3.2 Mida gran

Percolació per aresta



Percolació per node

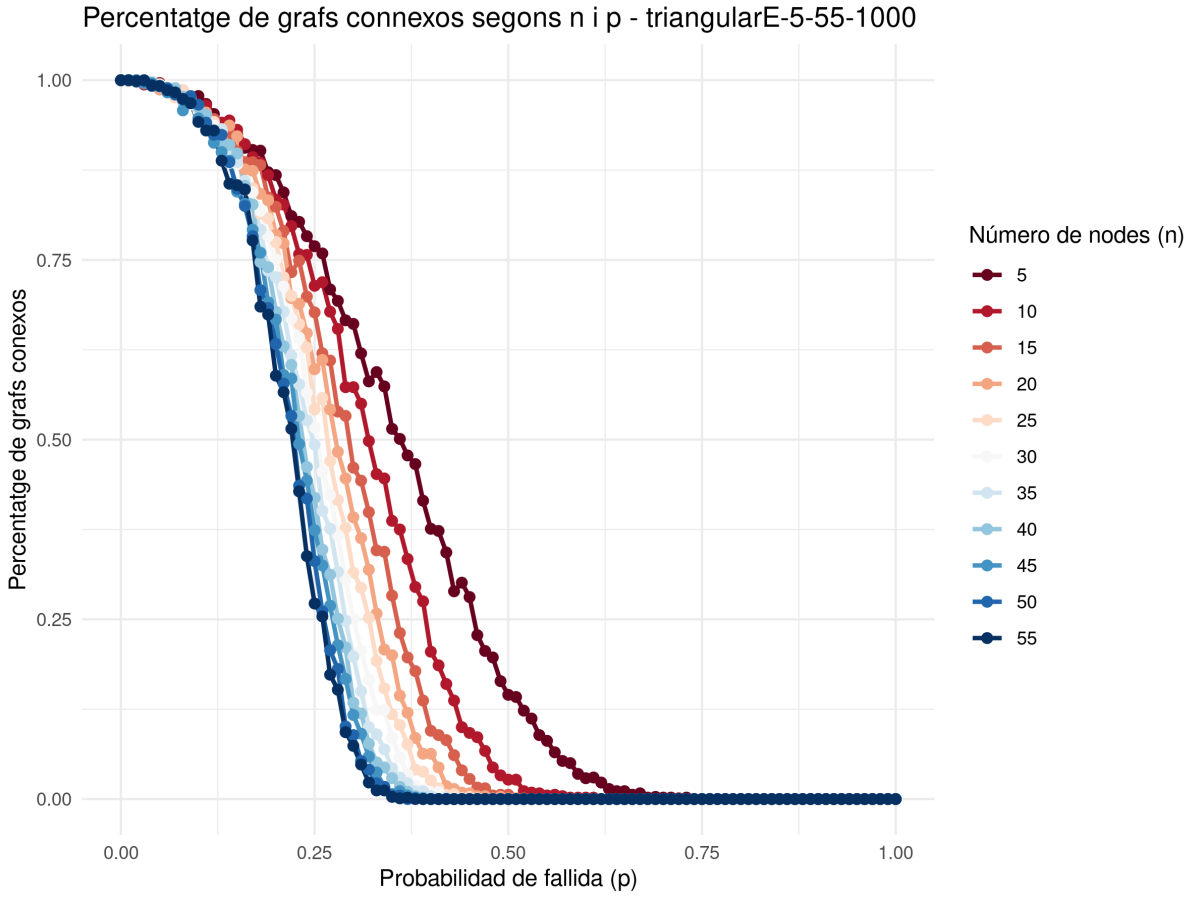


4.4 Graella triangular

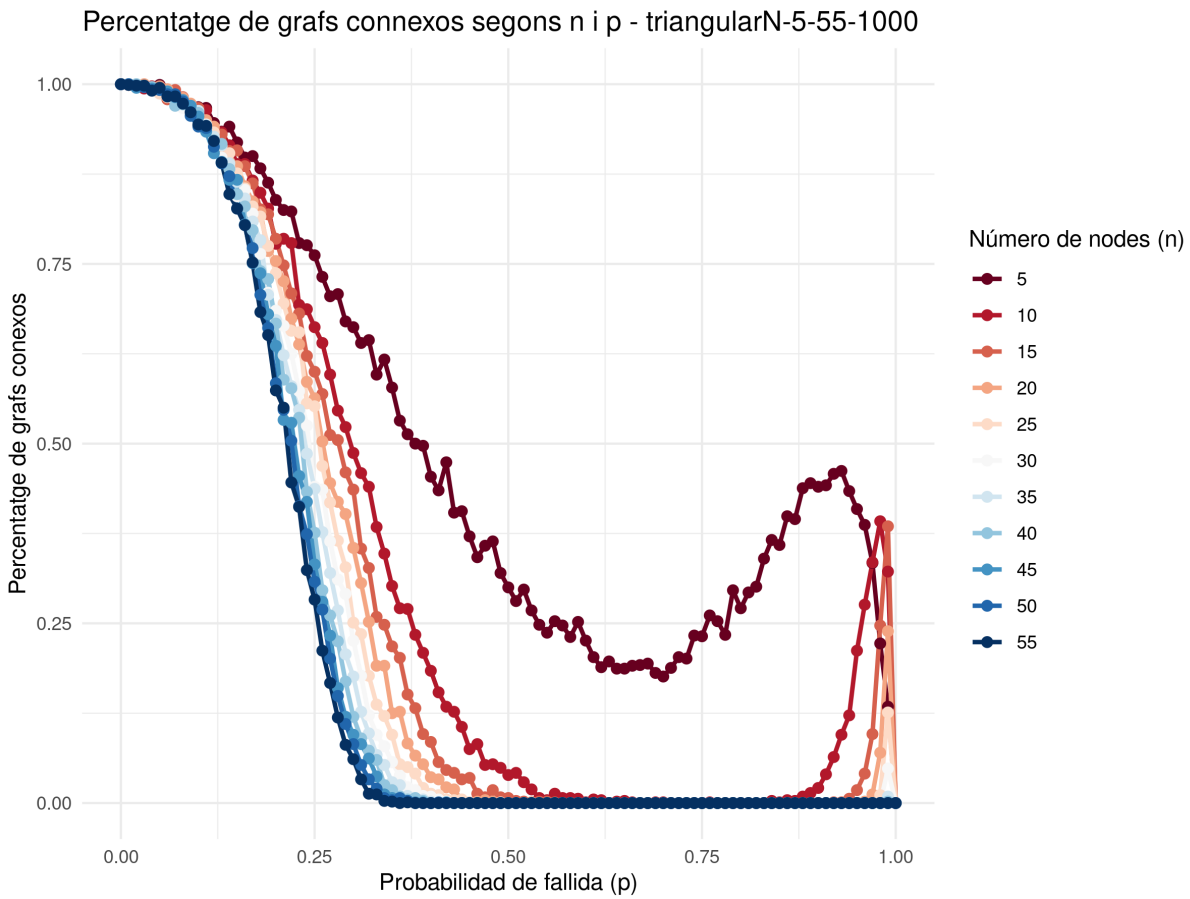
Ara estudiem la transició de fase en graelles triangulars. Per l'experimentació hem fet el mateix que en les graelles quadrades, estudiar la percolació per aresta i per node en grafs de mida petita i mida gran. En aquest cas, els grafs petits comencen amb 5 files fins a 55 files, amb increment de 5 files cada vegada, mentre que la mida gran son grafs de 50 a 150 files amb increments de 10 files.

4.4.1 Mida petita

Percolació per aresta

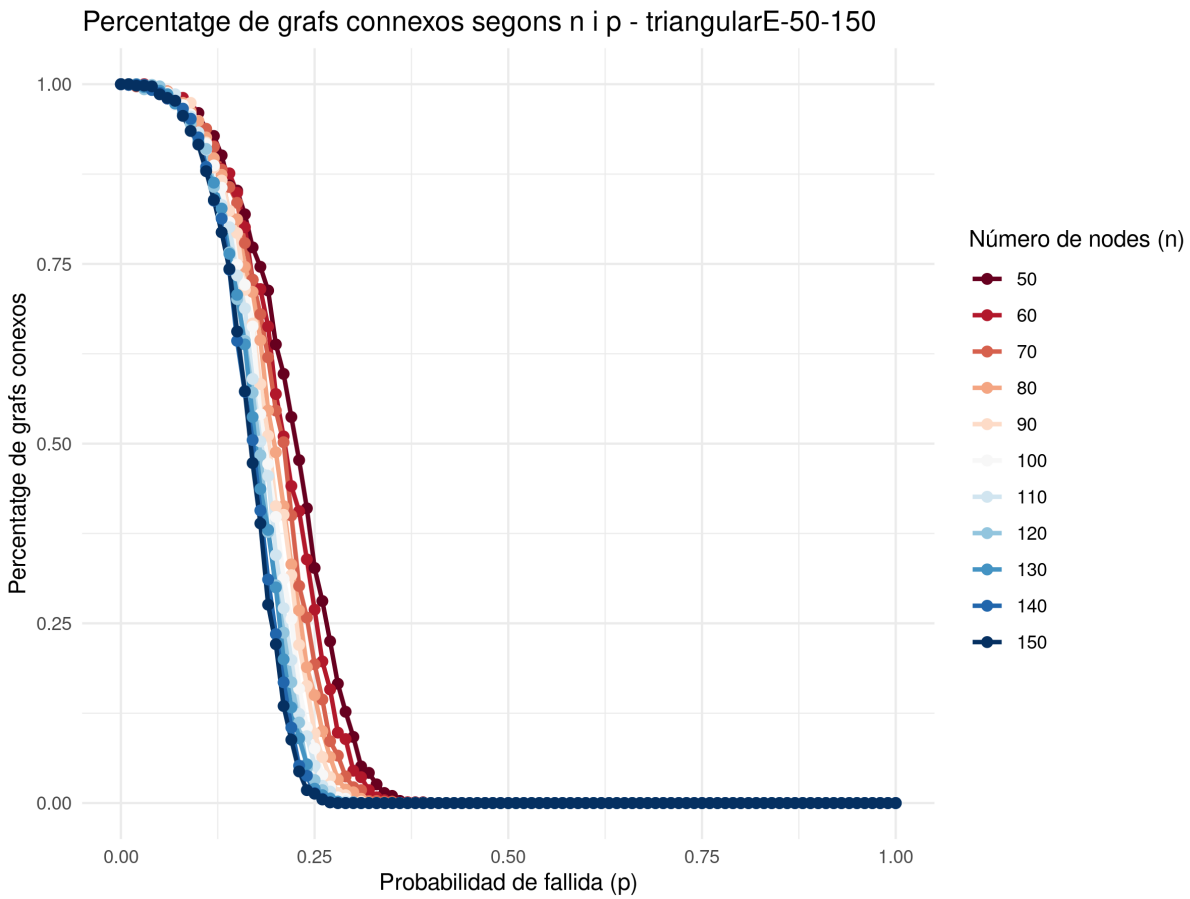


Percolació per node

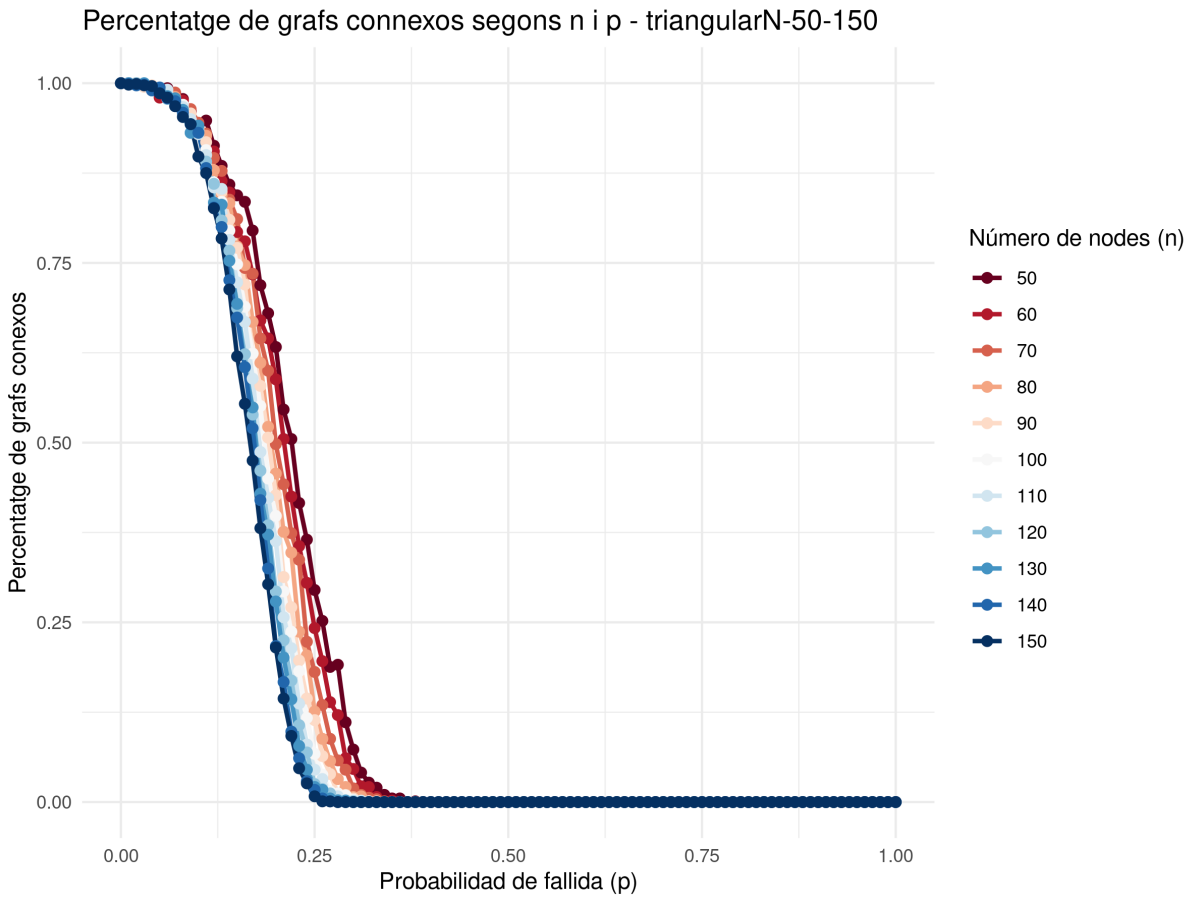


4.4.2 Mida gran

Percolació per aresta



Percolació per node



5 Conclusions

6 Bibliografia

- Wikipedia. *Model d'Erdős-Rényi*.

7 Annex