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 Graf

1.2 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}.

1.3 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.

1.4 Objectius de la Experimentació

2 Grafs Seleccionats

2.1 Erdős-Rényi

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

Inicialitzar el graf g amb n nodes
for $i = 0$ fins a $n - 1$ **do**
    for $j = i + 1$ fins a $n - 1$ **do**
        if rand01() $<\ p$ **then**
            Afegir una aresta entre i i j al graf g
        end if
    end for
end for
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 Square-Grid

2.3 Triangular-Grid

2.4 Random-Geometric

2.5 Barábasi-Albert

3 Algoritmes

3.1 Percolació per Nodes

3.2 Percolació per Arestes

3.3 Càlcul de Components Connexes

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 2 Descripció de l'experiment

```
Seleccionar opcions de configuració
Inicialitzar el generador de nombres aleatoris
Obrir l'arxiu CSV i escriure la capçalera
for n in range(MIN_NB_NODES, MAX_NB_NODES + 1, NB_NODES_STEP) do
    for p des de 0 fins a 1 amb pas 0.01 do
        Inicialitzar el comptador de grafs connexos
        for i = 0 fins a TRIES_PER_P do
            repeat
                Generar el graf seleccionat(p, n, params)
            until el graf és connex
            Aplicar percolació (per nodes o arestes) al graf
            if el graf percolat és connex then
                Incrementar el comptador de grafs connexos
            end if
        end for
        Escriure entrada resultant al CSV
    end for
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`.

5 Conclusions

6 Bibliografia

- Wikipedia. *Model d'Erdős-Rényi*.

7 Annex