Projecte d'Algorísmia

Transició de fase i components connexes en grafs aleatoris

Grau A Tardor Curs 2019-2020

Aleix Boné Ribó
Alex González Godoy
Alex Herrero Pons
Albert Mercadé Plasencia

Índex

1 Introducció

En aquest projecte teníem com a objectiu la investigació de les transicions de fase en algunes propietats dels models de grafs aleatoris Random Binomial Graph (`BRG`) i Random Geometric Graph (`RGG`). La connectivitat d'aquests models de grafs és la primera propietat que se'ns demanava explorar, a més, també se'ns requeria indagar sobre la mida esperada de la component connexa més gran al graf. En quant a les propietats extres que nosaltres em escollit per ampliar la nostra investigació, ens hem decantat per:

- Existència de camins i cicles Eulerians.
- Existència de cicles (Si el graf és o no un bosc).

Tant en les dos propietats demanades a l'enunciat com a les proposades per nosaltres hem trobat de transicions de fase. També hem investigat la transició de fase en l'existència d'arbres, però la probabilitat de que el graf generat sigues un arbre era tant baixa en `BRG` i `RGG` que no es podia apreciar transició de fase (era pràcticament 0 en tots els casos).

2 Implementació de la classe `Graph`

Per representar els grafs hem implementat una classe `Graph`[1] que utilitza llistes d'adjacències donat que són més barates en memòria i la implementació d'un `Breadth First Search (BFS)` menys costosa en comparació amb les matrius d'adjacència.

```
1  #ifndef GRAPH_H
2  #define GRAPH_H
3
4  #include <vector>
5  #include <list>
6  #include <utility>
7
8  enum { NOTHING, EULERIAN_PATH, EULERIAN_CYCLE };
9
10 class Graph {
11     std::vector<std::list<std::size_t> > AdjList;
12
13     public:
14
15     Graph(const std::size_t &n);
16
17     void addUndirectedEdge(const std::size_t& a, const std::size_t& b);
18     void addDirectedEdge(const std::size_t& a, const std::size_t& b);
19
20     const bool hasCycles() const;
21     const unsigned int EulerianCycleAndEulerianPath() const;
22     const std::pair <unsigned int, unsigned int> ConnectedComponents() const;
23     bool isConnected() const;
24
25     void print() const;
26
27     const std::list<std::size_t>& neighbors(const std::size_t& v) const;
28 };
29
30 #endif /* ifndef GRAPH_H */
```

Listing 1: Graph.h

Dins d'aquesta classe hem implementat la constructora, que inicialitza la llista d'adjacència, i també els següents mètodes:

[1] Implementat a src/graph.cpp

- `addUndirectedEdge`: donats els identificadors de dos vèrtexs a i b, crea una aresta no dirigida entre ells.
- `addDirectedEdge`: donats els identificadors de dos vèrtexs crea una aresta dirigida entre ells.
- `hasCycles`: retorna un booleà que indica si el graf te cicles. Està implementat utilitzant el mètode `BFS`.
- `EulerianCycleAndEulerianPath`: expliquem més endavant la seva utilitat e implementació.
- `ConnectedComponents`: expliquem més endavant la seva utilitat e implementació.
- `isConnected`: retorna un booleà que indica si el graf és connex. Està implementat amb un `BFS`.
- `neighbours`: retorna una llista que conté els veïns del vèrtex.
- `print`: imprimeix per pantalla la morfologia del graf. Utilitzat sobretot per fer tests i *debugar*.

3 Generació dels Grafs

3.1 Random Binomial Graphs

Per generar un Random Binomial Graph (`BRG`) ens hem basat en el model d'Erdős-Rényi [4, 5] que per un graf $G = (V, E)$ diu que

$$\forall u, v \in V, \quad (u, v) \in E \text{ amb probabilitat } p \tag{1}$$

Hem implementat l'algoritme prèviament descrit sota la funció `BRG`[2]. Per cada vèrtex veiem totes les possible arestes i seguint una distribució de Bernoulli (`bernoulli_distribution`[2]) amb probabilitat p decidim quines afegim i quines no.

3.2 Random Geometric Graphs

En un Random Geometric Graph (`RGG`)[3] per a tot vèrtex $v \in V$ se li assigna unes coordenades (x_v, y_v) aleatòries entre 0.0 i 1.0 i les arestes entre dos vèrtex s'afegeixen si la distancia euclidiana entre les coordenades assignades als vèrtex es menor a r, tal com es descriu a l'equació 2:

$$\begin{aligned} \forall u, v \in V, \quad (u, v) \in E \iff & \operatorname{dist}(u, v) \leq r \\ & \operatorname{dist}(u, v) = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2} \end{aligned} \tag{2}$$

La creació dels `RGG`s la fem sota la funció `RGG`[3], on assignem a cada vèrtex unes coordenades aleatòries amb una `uniform_real_distribution`[6]. Després per cada vèrtex iterem la resta i afegim una aresta quan la distància entre els dos vèrtex és menor o igual que el radi r.

[2]Implementada a src/graph_generator.cpp
[3]Implementada a src/graph_generator.cpp

4 Comput de dades i generació de gràfiques

4.1 Càlcul Adaptatiu

Al fer les primeres gràfiques ens vam adonar que la gran majoria de punts eren o bé 0, o 1 i la zona de transició de fase era sovint en un interval molt petit. Per tal d'evitar calcular punts innecessaris al fer les gràfiques però tenir una bona resolució a l'interval en que es produïa la transició de fase van decidir implementar un mètode de *sampling* adaptatiu[1][4].

El mètode utilitzat consisteix en dividir els valors de 0 a 1 en 11 punts (0, 0.1, 0.2 ...) i calcular per a cada un dels punts la mitjana de les 500 repeticions. Un cop feta la primera passada, mirem si hi ha dos valors consecutius en que la diferència es major a *dy* calculem el valor pel punt entremig dels dos de manera recursiva fins que la diferència sigui menor *dy* o s'arribi al limit de la recursivitat. Aquest mètode permet obtenir gràfiques que mostren la corba de transició de fase de manera clara sense haver de computar punts innecessaris del gràfic a les zones planes. Ajustant el valor de *dy* podem decidir la resolució de la gràfica resultant.

Un cop compilat el programa (amb `make`), la binària `main` dins de *bin* calcula utilitzant el mètode descrit les dades per les diferents propietats que estudiarem. Es crida de la següent manera:

```
./bin/main tipus_graph propietat N repeticions dy
```

Descripció dels paràmetres:

- `tipus_graph` es el tipus de graf aleatori (`BRG` o `RGG`)
- `propietat` es la propietat a calcular (cicle, esConnex, eulerianCycle, eulerianPath, midaCompConMax, numCompCon)
- `N` es la mida del graf
- `repeticions` es el nombre de repeticions per cada punt (vam utilitzar 500)
- `dy` es el valor *dy* descrit anteriorment (vam utilitzar 0.1 o 0.01 en funció del tipus de dada)

4.2 Generació de gràfiques

Per generar les gràfiques a partir de les dades calculades, utilitzem les llibreries *matplotlib* i *numpy* de *Python 3*. El codi utilitzat per generar els gràfics individuals es troba a *plot.py*. L'*script* de bash *makePlots.sh* automatitza tot el proces de calcul i generació de les gràfiques. (`bash makePlots.sh 10 25 50 60 80 100 150`)

[4]Implementat a src/adaptative_calc.cpp

5 Experiments

5.1 Probabilitat de ser connex

En aquest primer experiment la nostra hipòtesi és que sí que trobarem una transició de fase en ambdós casos. En el cas del `BRG` a mesura que augmenta la probabilitat p de que existeixin arestes entre els vèrtex té una certa lògica que augmenti també la probabilitat de ser connex. D'altra banda, en el cas del `RGG` com més gran és r més arestes tindrà cada vèrtex i per tant més alta la probabilitat de que el graf sigui connex.

```
83  // retorna nombre de components connexes i mida de la component connexa gegant.
84  const std::pair <unsigned int, unsigned int> Graph::ConnectedComponents() const {
85      std::vector<bool> visited(AdjList.size(), false);
86
87      unsigned int count = 0;
88      unsigned int max = 0;
89      size_t next = 0;
90
91      std::queue<size_t> Q;
92
93      while (next < AdjList.size()) {
94
95          ++count;
96
97          unsigned size = 0;
98
99          Q.push(next);
100
101         // BFS
102         while (!Q.empty()) {
103             size_t v = Q.front();
104             Q.pop();
105
106             if (visited[v]) continue;
107
108             visited[v] = true;
109             ++size;
110
111             for (const size_t &u : this->neighbors(v))
112                 if (!visited[u]) Q.push(u);
113         }
114
115         if (size > max) max = size;
116
117         // Busquem seguent vertex no visitat (nova component)
118         for (;next < AdjList.size() && visited[next]; ++next);
119     }
120
121     return std::make_pair(count, max);
122 }
```

Listing 2: Funció de ConnectedComponents a graph.cpp

Per poder comprovar la hipòtesi i saber quantes components connexes (cc) té el `Random Graph` generat, hem escrit la funció `ConnectedComponents`, la qual mitjançant l'algoritme de cerca `BFS` analitza el graf en busca de cc. Com es veu en el codi, primerament s'inicialitza un vector de booleans per saber quins vèrtexs ja han estat visitats. Començant pel primer vèrtex i amb l'ajuda d'una cua, es recorren els vèrtexs adjacents no visitats encara mentre s'afegeixen a la cua els veïns d'aquests. Durant la cerca, es manté un comptador de cc trobades que s'augmenta quan ja s'han visitat tots els vèrtex de la cc actual i encara hi ha veïns per visitar. Un cop la cua ja està buida, és a dir, no hi ha més vèrtexs de la cc per visitar, es busca el següent vèrtex no visitat per analitzar una nova component. Finalment retorna un *pair* amb la quantitat de cc que hi ha i la mida de la major. El cost d'aquest algoritme és $\mathcal{O}(n+m)$ ja que simplement fem ús d'un `BFS`.

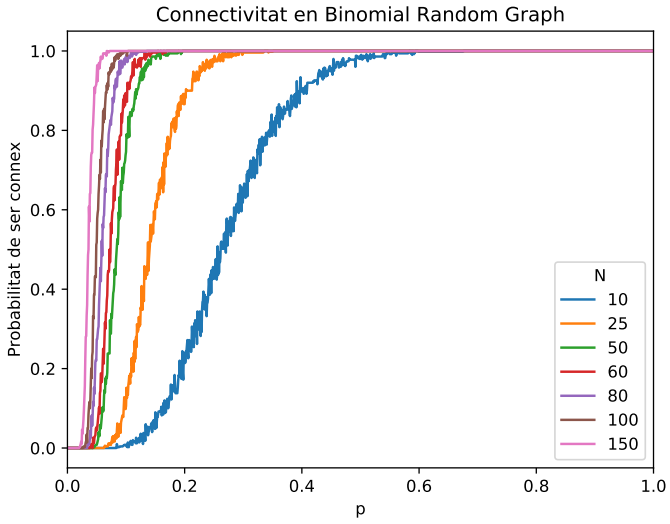

Figura 1: Connectivitat de `BRG` per diferents valors de n

A l'hora d'analitzar els resultats i poder constatar la transició de fase hem plasmat les dades obtingudes en un gràfic on es representa la probabilitat de ser connex de `BRG` per diferents valors de n i per tota possibilitat p. A la Figura 1 es pot apreciar una transició de fase més marcada per valors elevats de N a probabilitat de p propera a 0.05. De manera que podríem dir amb certa certesa que la hipòtesi és certa.

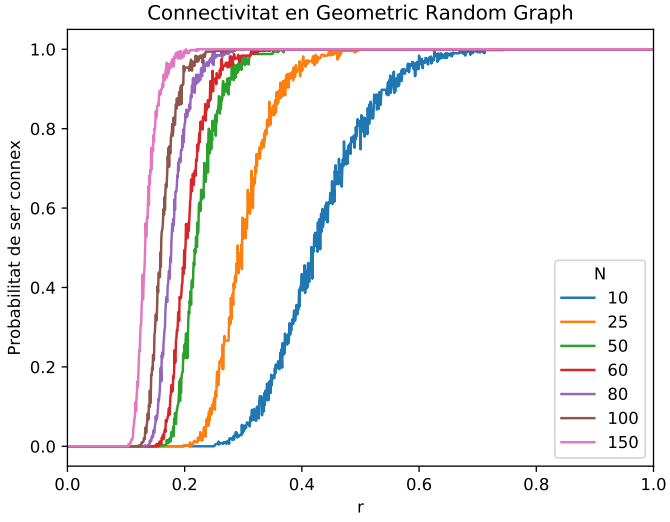

Figura 2: Connectivitat de `RGG` per diferents valors de n

En el cas del `RGG`, com es pot veure a la Figura 2, representem la probabilitat de que el graf sigui connex respecte el radi r per a varis valors de N. En aquest cas també apreciem una transició de fase bastant clara que confirma la nostra hipòtesis. A més, semblant al cas del `BRG`, veiem que la transició és més lenta a mesura que el valor de N augmenta. Té sentit que sigui així donat que quants més vèrtex hi han al graf més fàcil és que que hi hagi arestes entre ells connectant-los. En el cas de $N = 150$ la transició succeeix molt ràpidament entre els valors de 0.1 i 0.17 de r. En canvi per $N = 10$ la transició ocorre

entre $r = 0.25$ i $r = 0.6$. Conforme aquests resultats deduïm que la nostra hipòtesi concorda amb ells.

5.2 Mida esperada CC més gran

Per aquest experiment, pels mateixos motius que la connectivitat i ja havent confirmat que era l'existència de la transició, vam pensar que seria coherent que la mida esperada de la CC més gran també tingués transició de fase, tant en `BRG` com en `RGG`.

El codi que hem utilitzat en aquest cas ha sigut el mateix que per la probabilitat de ser connex mostrat més amunt en el Listing 2, però hem afegit dues variables ($size$, max) per tal de comprovar quina component és la més gran. Per tant el cost també és $\mathcal{O}(n+m)$.

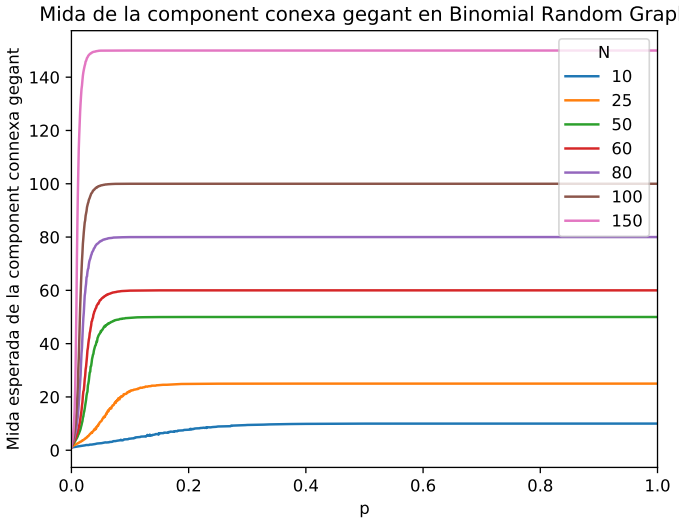

Figura 3: Mida esperada de la CC més gran de `BRG` per diferents valors de n

Amb els resultats obtinguts es pot veure clarament una transició de fase propera a p=0.05, la qual era en certa mida previsible després dels resultats de la connectivitat de `BRG` per a diferents valors de n, ja que la mida d'una component connexa gegant és inversament proporcional a la quantitat de components connexes. Arrel d'això podem concloure que la nostra hipòtesi també es podria complir en aquest cas.

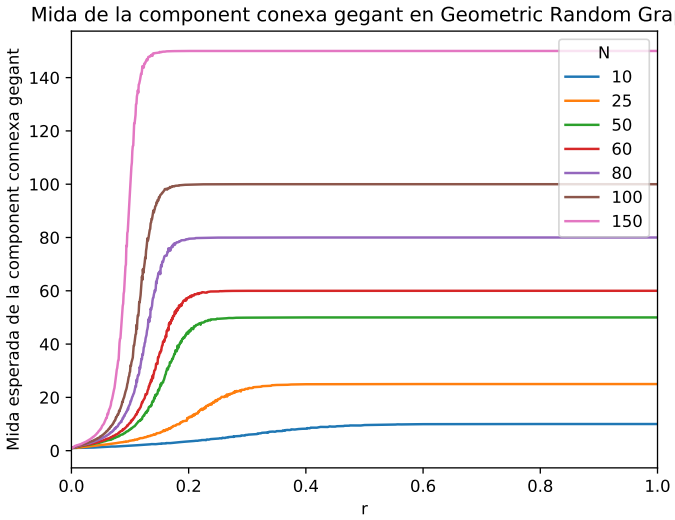

Figura 4: Mida esperada de la CC més gran de `RGG` per diferents valors de n

En el cas de `RGG` els resultats representats ens ajuden a veure una transició de fase ben marcada a p=*0.1* la qual correspon a la obtinguda en el anàlisi de la connectivitat de `RGG` per diferents valors de n. Com esmentat anteriorment, aquests resultats venen justificats pel fet de la proporcionalitat inversa entre el nombre de *cc* i la mida de la component connexa gegant. En base a aquests resultats, podem dir amb certa seguretat que la nostra hipòtesi és certa.

5.3 Camí i cicle Eulerià

En aquest tercer experiment hem volgut comprovar si la probabilitat de tenir un camí o un cicle Eulerià tindria transició de fase. Un camí Eulerià és aquell que recorre totes les arestes del graf passant només un cop per cadascuna d'elles. El cicle Eulerià vindria a ser un camí Eulerià que a més comença i acaba al mateix vèrtex. Vam escollir aquesta propietat perquè ens semblava més arbitrària i que per tant no tindria transició de fase.

Per determinar si un graf té un cicle Eulerià o un camí Eulerià hem creat la funció `EulerianCycleAndEulerianPath` que es mostra en el Listing 3. Primerament utilitzant la funció `ConnectedComponents` comprova que tots el vèrtex amb grau $g > 0$ pertanyen a una única Component Connexa, si és el cas simplement mirant el nombre de vèrtexs amb grau imparell podem classificar si tenen un camí (dos vèrtexs imparells) o si tenen un cicle, i per tant també un camí (cap vèrtex imparell).

```
8  enum { NOTHING, EULERIAN_PATH, EULERIAN_CYCLE };
```

```
60 const unsigned int Graph::EulerianCycleAndEulerianPath() const {
61     unsigned int nVertex = AdjList.size();
62
63     std::pair<unsigned int, unsigned int> info = ConnectedComponents();
64     unsigned int nCC = info.first;
65     unsigned int max = info.second;
66
67     // all of its vertices with nonzero degree belong to a single connected component
68     bool singleCC = (nCC + max - 1) == nVertex;
69
70     if (singleCC) {
71         int oddVertex = 0;
72         for (unsigned int i = 0; i < nVertex; ++i)
73             if (AdjList[i].size() % 2 != 0)
74                 ++oddVertex;
75
76         if (oddVertex == 0) return EULERIAN_CYCLE;
77         if (oddVertex == 2) return EULERIAN_PATH;
78     }
79
80     return NOTHING;
81 }
```

Listing 3: Funció de EulerianCycleAndEulerianPath en graph.cpp

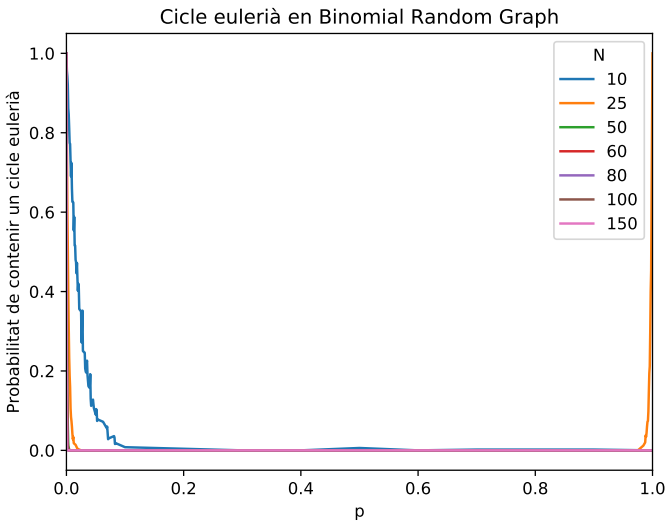

Figura 5: Eulerian Cycle de `BRG` per diferents valors de n

Com es pot veure a la Figura 5, es podria considerar l'existència d'una transició de fase per valors de p propers a 0 ja que en aquell moment tot vèrtex amb $g = 0$ es considera cicle Eulerià. Però la probabilitat de contenir un cicle Eulerià baixa dràsticament ja que s'han de donar les condicions que hi hagi un cicle i que a més aquest sigui Eulerià. Finalment, es pot veure que amb valors de p pròxims a 1, la probabilitat que un graf $N = 25$, contingui un cicle Eulerià creix fins a 1 un altre cop. Això es deu a que tenir $p = 1$ obtens un graf complet (K_N) el qual, si el nombre de vèrtexs, N és imparell, tindrà un cicle Eulerià, ja que tots els vèrtexs tenen grau $N-1$ que és parell (al ser imparell N). Segons els resultats obtinguts i l'existència de la transició de fase podem constatar que la nostra hipòtesi en aquest cas no es compleix.

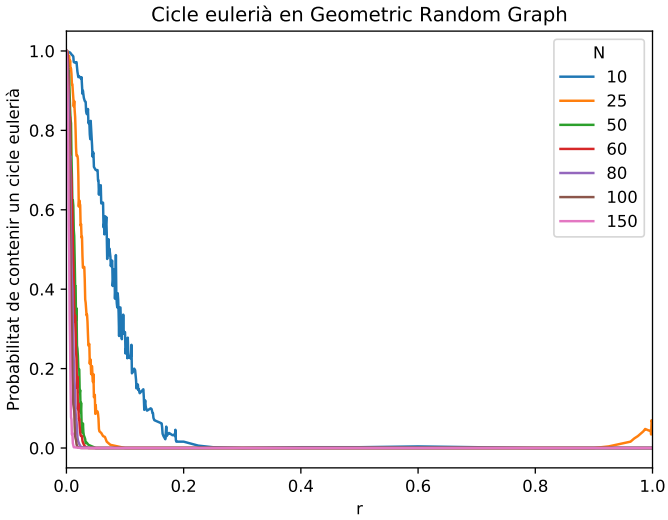

Figura 6: Eulerian Cycle de `RGG` per diferents valors de n

En el cas de `RGG`, la transició de fase està present per a valors de p propers a 0, però en aquest cas, a diferència del `BRG`, no està tant marcada quan el nombre de vèrtexs és petit. Igual que en el cas de `BRG`, els vèrtexs amb grau 0 es consideren cicles Eulerians. De nou, amb valors molt elevats de p i amb nombre imparell de vèrtexs es pot veure que la probabilitat que contingui un cicle Eulerà torna a créixer. Tot i no estar tant marcada la transició de fase també es present, per tant podem dir que la nostra hipòtesi tampoc es compleix.

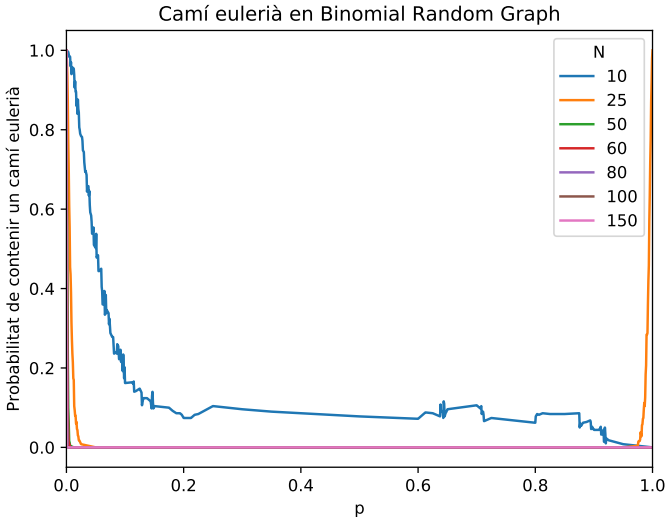

Figura 7: Eulerian Path de `BRG` per diferents valors de n

Amb els resultats obtinguts a l'hora de trobar un camí Eulerià podem veure a la Figura 7 que la transició de fase no està tant marcada com en el de la Figura 5 ja que en aquest cas només s'ha de complir la condició que hi hagi un camí, no que hi hagi un cicle. A més, es pot comprovar que al tenir un graf amb un nombre imparell de vèrtexs ($N = 25$) i $p = 1$ obtens un graf complet el qual sempre tindrà un cicle Eulerià i per tant un camí

Eulerià. Encara amb les diferències amb la probabilitat de contenir un cicle Eulerià, la transició de fase clarament es produeix, denegant la nostra hipòtesi.

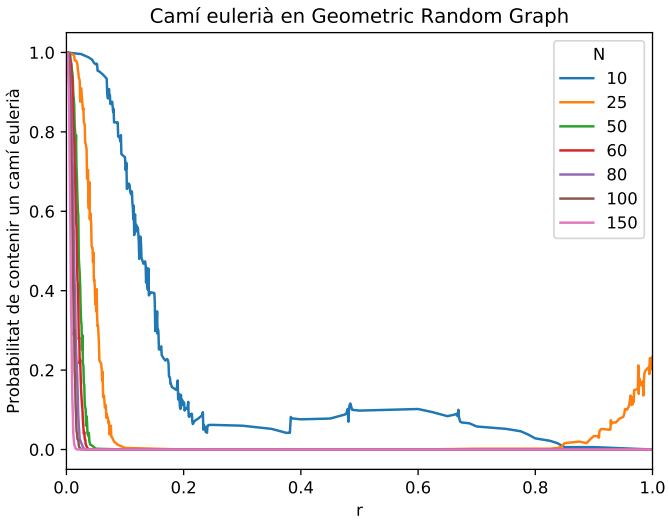

Figura 8: Eulerian Path de `RGG` per diferents valors de n

Finalment, amb el `RGG` tornem a trobar una transició de fase no tant clara com amb `BRG` però tot i així present per valors de p propers a 0 pel mateix motiu que a la Figura 6. També es pot veure que per valors petits de N, com es el cas de $N = 10$ la probabilitat d'haver-hi un camí Eulerià va variant però no acaba de ser 0 fins valors de p pròxims a 0. I a més en cas de $N = 25$ amb p proper a 1 torna a pujar la probabilitat de contenir un camí Eulerià com a la Figura 6. Segons els resultats obtinguts i la transició de fase representada, podem concloure que la nostra hipòtesi no es compleix en aquest cas.

5.4 Probabilitat de contenir un cicle

L'últim experiment que hem realitzat és veure si la probabilitat de que un graf contingui un cicle té transició de fase o no. Abans de realitzar l'experiment, havent vist que el cicle Eulerià si que tenia transició de fase vam anticipar que en aquest cas també en tindria i més remarcada tant pel `BRG` com pel `RGG`.

Per comprovar si el graf conté algun cicle hem creat la funció `hasCycles` que recorre el graf amb un `BFS`, com prèviament hem explicat, i si en la mateixa component connexa torna a visitar un mateix vèrtex retorna que ha trobat un cicle.

```cpp
const bool Graph::hasCycles() const {
    std::vector<bool> visited(AdjList.size(), false);

    size_t next = 0;

    std::queue<size_t> Q;

    while (next < AdjList.size()) {

        Q.push(next);

        // BFS
        while (!Q.empty()) {
            size_t v = Q.front();
            Q.pop();

            // Si estem tornant a visitar un vertex -> hi ha cicle
            if (visited[v]) return true;

            visited[v] = true;

            for (const size_t &u : this->neighbors(v))
                if (!visited[u]) Q.push(u);
        }
        // Busquem seguent vertex no visitat (nova component)
        for (;next < AdjList.size() && visited[next]; ++next);
    }

    return false;
}
```

Listing 4: Funció de hasCylces

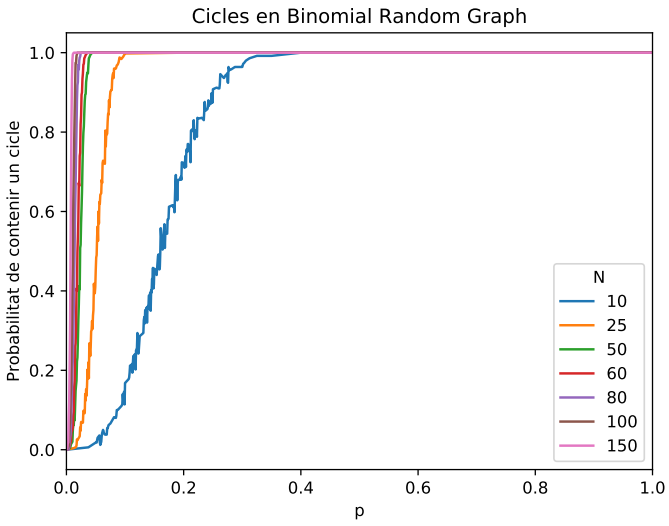

Figura 9: Probabilitat de tenir un cicle de BRG per diferents valors de n

En el cas del BRG veiem a la Figura 8 com la transició de fase és evident. Per les N més grans la transició és molt ràpida i succeeix amb probabilitats molt baixes. Aquest comportament s'explica per que a mesura que tenim més vèrtex, encara que la probabilitat de tenir una aresta entre dos vèrtex sigui baixa, també incrementa la quantitat d'arestes i per tant augmenta la probabilitat de que apareguin cicles. En canvi, com es veu pel cas de $N = 10$, per N més petites és més difícil que això passi. Per tant, segons els resultats obtinguts, la nostra hipòtesi es confirma pel BRG.

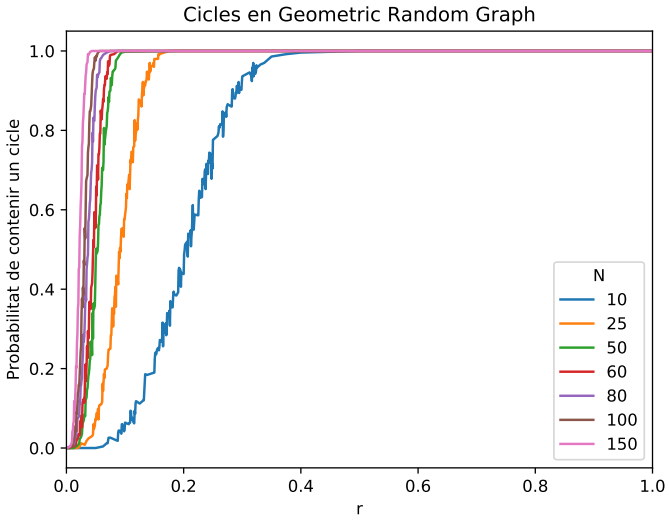

Figura 10: Probabilitat de tenir un cicle de `RGG` per diferents valors de n

Pel `RGG` el resultat és similar al `BRG` i s'aprecia una transició de fase clara. Entre les N més elevades les transicions no són tant semblants i es distingeix més l'efecte de variar N, com més vèrtexs tenim més petit és el radi necessari per que la transició aparegui. Com en el `BRG`, en el cas de $N = 10$ l'aparició de la transició respecte al radi r és clarament molt més menys pronunciada.

6 Possibles futurs experiments

Com a possibles experiments havíem fet, aprofitant la probabilitat de si un graf conté un cicle, la probabilitat de que el graf sigués un arbre (no tenir cicles i ser connex). Però veient els resultats finalment no l'hem volgut exposar, ja que per que es doni que el graf sigui un arbre ha de tenir un nombre determinat d'arestes i a més no tenir cap cicle, cosa que generant el graf de forma aleatòria és molt poc probable, i en els nostres experiments gairebé cap era un arbre.

A tots els experiments només hem usat una N imparella ($N = 25$) i seria interessant, sobretot en el cas de camins i cicles Eulerians, haver inclòs més N imparells. També havíem pensat en veure si hi ha transició de fase en la probabilitat de que hi hagi un camí o cicle Hamiltonià.

A més també havíem considerat indagar la existència de transició de fase de que els grafs siguin bipartits però finalment ens vam decantar per les propietats explicades als experiments.

7 Conclusions

A partir dels resultats obtinguts podem concloure que en major o menor mesura la transició de fase és present a totes les propietats estudiades. Observem també que com més vèrtex té el graf, més pronunciada es la seva transició de fase i es produeix per valors més petits de r o p. En el cas de camí i cicle Eulerià podem apreciar una transició de fase inicial clara, però a mesura que p i r s'apropen a 1, quan tenim un nombre imparell de vèrtex, com és el cas de $N = 25$, observem un pronunciat canvi de tendència. Això és

degut a que tots els grafs complets K_N, amb N imparell, són cicle Eulerià (tots els vèrtex tenen grau parell). Per tant hem après que pels grafs aleatoris `BRG` i `RGG` les transicions de fase són comunes a moltes propietats del graf, inclús al cas del camí i cicle Eulerià on creiem que succeiria.

Referències

[1] *Adaptive function plotting*. URL: https://yacas.readthedocs.io/en/latest/book_of_algorithms/basic.html#adaptive-function-plotting.

[2] *bernoulli_distribution - C++ Reference*. URL: http://www.cplusplus.com/reference/random/bernoulli_distribution/.

[3] Díaz, J, Mitsche, D i Pérez, X. *On the Connectivity of Dynamic Random Geometric Graphs **. Inf. tèc. 2007.

[4] Erdős, P., Erdős, P. i Rényi, A. «On the Evolution of Random Graphs». A: *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES* (1960), pàg. 17 - 61. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.5943.

[5] Erdős, P. i Rényi, A. «On Random Graphs. I». A: *Publicationes Mathematicae* 6 (1959), pàg. 290 - 297. URL: http://www.renyi.hu/~p_erdos/1959-11.pdf.

[6] *uniform_real_distribution - C++ Reference*. URL: http://www.cplusplus.com/reference/random/uniform_real_distribution/.