Transició de fase en propietats de grafs

Grau A

Q1 Curs 2021-2022

Gil Ralló

Pol Casacuberta

Alexandru-Ilie Popa

Índex

1. Descripció i Anàlisis del Problema

L'objectiu d'aquest projecte era fer un estudi experimental de la transició de fase de propietats de grafs en processos de percolació.
La representació d'aspectes de la realitat per grafs és molt utilitzada. I en la realitat pot passar que el que és representat per un vèrtex o una aresta falli, llavors el graf s'ha de veure modificat. Això és del que tracta bàsicament el procés de percolació d'un graf, aquest procés està definit per un paràmetre $q \in [0, 1]$ que representa la probabilitat de no fallida. De tal manera que en aplicar un procés de percolació sobre un graf G i una *q* obtindrem un G_q en el que cada node $u \in V(G)$ continuarà sent a $V(G_q)$ amb una probabilitat de *q* o hi haurà set eliminat del graf G, i , per tant, no pertanyerà a $V(G_q)$ amb una probabilitat de (1-*q*). El procés de percolació es pot aplicar tant per nodes (site percolation) com per arestes (bond percolation). En el cas de la percolació per nodes, en eliminar un vèrtex també s'eliminen les arestes adjacents a aquest i en la percolació per arestes només s'elimina l'aresta, per tant, en la percolació per arestes $V(G) = V(G_q)$.

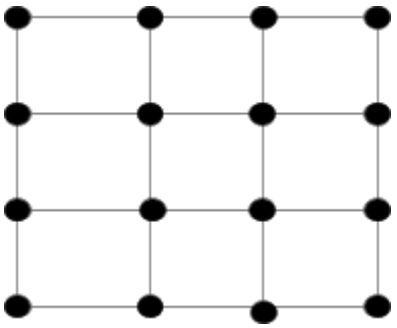
Una vegada fixat això havíem d'estudiar l'existència o no de transició de fase en processos de percolació sobre 2 propietats. Una transició de fase es dona quan tenim una q_π, que per a valors superiors es compleix amb alta probabilitat una propietat Π, i per a valors inferiors no es compleix amb alta probabilitat. Per tant, per a $q < q_\pi$ tindrem que els G_q no compleixen la propietat i per a $q_\pi < q$ els G_q sí que la compleixen. Amb els experiments haviem de comprovar si es presentava o no una transició de fase al voltant d'una q_π.

Les dues propietats sobre les quals havíem de comprovar si existia aquesta transició de fase eren que el graf fos connex i que les components connexes fossin complexes.
Un graf és connex quan existeix un camí des de qualsevol vèrtex $u \in G$ a qualsevol $v \in G$ sent $u \neq v$. És a dir, que des d'un vèrtex es pot arribar a qualsevol altre vèrtex del graf.

Una component connexa, o un graf connex, no és complex quan és un arbre o un graf unicíclic, que és el mateix que dir que perquè una component connexa sigui complexa ha de tenir 2 o més cicles.

Tot aquest estudi l'havíem de dur a terme sobre 3 tipus diferents de grafs:

Graf graella: és un graf connex de n x n vèrtexs col·locats en forma de graella quadrada on cada un és adjacent amb els 2, 3 o 4 vèrtexs situats al seu voltant. Com es pot observar a la figura següent amb n = 4:



Binomial Random Graph: és un graf representat com G(n, p), on *n* és el nombre de vèrtexs i $p \in [0, 1]$ és la probabilitat que una aresta entre 2 qualsevol d'aquests *n* vèrtexs hi sigui, independentment de les altres. Com més pròxima a 1 sigui la *p* més arestes tindrà el graf normalment i com més pròxima a 0 menys, sent el graf G(n, 0) el graf amb *n* vèrtexs i cap aresta entre aquests i el graf G(n, 1) el graf complet amb *n* vèrtexs. Com més arestes tingui el graf, és a dir, com més gran sigui la *p*, més possibilitats tindrà de ser connex.

Random Geometric Graph: és un graf representat com G(n, r), on *n* és el nombre de vèrtexs i *r* és la distància a partir de la qual 2 vèrtexs deixaran de ser adjacents. En aquest tipus de grafs els vèrtexs estan distribuïts en un espai de coordenades entre 0 i 1: una línia de longitud 1 si l'espai és d'1 dimensió, una pla de costat 1 si l'espai és de 2 dimensions o un cub de costat 1 si l'espai és de 3 dimensions. Es té en compte la distància euclidiana entre vèrtexs.

$$d(x,y) = ||x-y||_2 = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}.$$

El valor màxim que podrà adquirir *r* serà la distància entre els 2 punts més llunyans de l'espai en el qual es tracti: 1 si és un espai d'1 dimensió, $\sqrt{2}$ si és un espai de 2 dimensions o $\sqrt{3}$ si és un espai de 3 dimensions.
Dos vèrtexs seran adjacents si la seva distància euclidiana és menor que *r*. Per tant, com més gran sigui *r*, vèrtexs situats a més distància podran ser adjacents i per tant hi haurà més arestes, i com més pròxima a 0 sigui *r,* menys vèrtexs podran ser adjacents i per tant hi haurà menys arestes. Com més arestes tingui el graf, és a dir, com més gran sigui *r*, més possibilitats tindrà de ser connex.

El llenguatge que hem fet servir és python, ja que té llibreries molt útils a l'hora de treballar amb grafs

2. Algorismes

2.1 Components Connexes i Propietats

Per saber si un graf és connex hem fet servir la funció *is_connected(graf)* de la llibreria networkx. Si ho haguéssim d'implementar, bàsicament és fer un DFS i comprovar si al final s'han visitat tots els vèrtexs. En cas que no s'hagin visitat tots, tots els visitats pertanyen a la mateixa component connexa. Per obtenir les components connexes hem fet servir la funció *connected_components(graf)* de la llibreria networkx, que ens retorna un array amb totes les components connexes.

Per comprovar la segona propietat, que totes les components connexes siguin complexes hem fet la següent funció:

```
def complex_connected_components(g):
      """
      Returns true if all the connected components are complex
     (have at least two cycles)

      :param g: Graph to be percolated
      :return: True if all the CC are complex
      """
      b = True
      for c in nx.connected_components(g):
      h = g.subgraph(c)
      b &= len(nx.cycle_basis(h)) > 1
      if not b:
            return b
      return b
```

Aquesta funció té cost $O(n(n+m)n^{\gamma})$, sent $O(n(n+m))$ el cost de la funció connected_components(g), com a molt un bfs (de cost $O(n+m)$) per cada vèrtex de g (graf amb n vèrtexs i m arestes). El cost de cycle_basis és $O(n^{\gamma})$ on $2 <= \gamma <= 3$.

2.2 Generació de Grafs Aleatoris i del Grafs Graella

Per la creació dels grafs graella hem fet servir la següent funció:

2.2.1 Graella

```
def graella_nxn(n):
      """
      Generate a graella graph

      :param n: number of nodes to make an NxN graph
      :return: Returns a "graella" graph
      """
      g = nx.Graph()
      for i in range(n * n):
      g.add_node(i)
      # Arestes horitzontals
      for i in range(n):
      for j in range(n - 1):
            g.add_edge((i * n) + j, (i * n) + j + 1)
      # Arestes verticals
      for i in range(n - 1):
      for j in range(n):
            g.add_edge((i * n) + j, ((i + 1) * n) + j)
      return g
```

Aquesta funció té cost $\Theta(n^2)$, sent *n x n* el nombre de vèrtexs del graf graella resultant.

Tant per la creació dels binomial random graphs com pels random geometric graphs hem fet servir les funcions *binomial_graph(Nnodes, p, dirigit)* i *random_geometric_graph(Nnodes, r)* de la llibreria networkx que crea els 2 tipus de grafs amb els seus Nnodes i les seves *p* i *r* corresponents.

El cost de la funció binomial_graph es $O(n^2)$, sent *n* el nombre de vèrtexs del graf. El cost de la funció is_connected es $O(n + m)$, ja que és un bfs.

2.2.2 Binomial Random Graphs

```
def binomial_graph():
      """
      Generates a plot which shows how probable it is that a binomial
      generated graph is connected
      """
      if not os.path.isdir(directory_path + "/binomial_graph"):
      os.makedirs(directory_path + "/binomial_graph")
      # We try for every probability 10 times Ex: if two times the graph
        is connected then we have a 20%
      times = 10
      # probability that it is indeed connected
      nplot = 0
      node_values = [10, 20, 50, 100, 500, 1000, 2000, 5000, 10000]
      for Nnodes in tqdm(node_values, desc="Nodes"):
          numbers_x = []
          numbers_y = []
          for prob in tqdm(np.linspace(0, 1, 11), desc="Probability",
                           leave=False
                          ):
            n_connected = 0
            for _ in tqdm(range(times), desc="Times", leave=False):
                  bi_graph = nx.binomial_graph(Nnodes, prob, directed=0)
                  if nx.is_connected(bi_graph):
                  n_connected = n_connected + 1
            p_connected = n_connected / times
            numbers_x.append(prob)
            numbers_y.append(p_connected)
      connected_plot(numbers_x, numbers_y, "Probability that an edge is
                     created", nplot, Nnodes,"/binomial_graph/plots/"
                    )
      nplot += 1
```

2.2.3 Random Geometric Graphs

```
def random_geometric_graph():
      """
      Generate a plot about how connected a graph is given a radius to
      generate the graph
      """
      if not os.path.isdir(directory_path + "/random_geometric_graph"):
      os.makedirs(directory_path + "/random_geometric_graph")
      times = 10
      nplot = 0
      node_values = [10, 20, 50, 100, 500, 1000, 2000, 5000, 10000]
      for Nnodes in tqdm(node_values, desc="Nodes"):
          numbers_x = []
          numbers_y = []
          for radius in tqdm(np.linspace(0, math.sqrt(2), 11),
                             desc="Radius", leave=False
                            ):
            n_connected = 0
            for _ in tqdm(range(times), desc="Time", leave=False):
                  geo_graph = nx.random_geometric_graph(Nnodes, radius)
                  if nx.is_connected(geo_graph):
                  n_connected = n_connected + 1
            p_connected = n_connected / times
            numbers_x.append(radius)
            numbers_y.append(p_connected)
      connected_plot(numbers_x, numbers_y, "Radius where edges are
                     created between nodes", nplot, Nnodes,
                     "/random_geometric_graph/plots/"
                    )
      nplot += 1
```

El cost de la funció random_geometric_graph és O(n), sent *n* el nombre de vèrtexs del graf. El cost de la funció is_connected és *O(n + m)*, ja que és un bfs.

2.3 Procés de Percolació

Per implementar la percolació simplement hem recorregut tots els nodes o les arestes del graf (depenent de si fem una percolació per nodes o per arestes) i hem eliminat el graf o l'aresta amb una probabilitat (1-p).

2.3.1 Percolació per Nodes (Site Percolation)

```
def node_percolation(g, prob):
      """
      For all nodes in g if a random generated number is greater than
      given value p then we remove the node

      :param g: Graph to be percolated
      :param prob: If p is < random then node is removed from the graph
      :return: Percolated graph
      """
      for i in range(g.number_of_nodes()):
      if random.random() > prob:
            g.remove_node(i)
      return g
```

Aquesta funció té cost Θ(n) sent *n* el nombre de vèrtexs del graf.

2.3.2 Percolació per Arestes (Bond Percolation)

```
def edge_percolation(g, prob):
      """
      For all edges in g if a random generated number is greater than
      given value p then we remove the edge

      :param g: Graph to be percolated
      :param prob: If p is < random then edge is removed from the graph
      :return: Percolated graph
      """
      for i in g.edges():
      if random.random() > prob:
            g.remove_edge(*i)
      return g
```

Aquesta funció té cost Θ(m) sent *m* el nombre d'arestes del graf.

Després hem aplicat aquest procés de percolació sobre els 3 models de grafs que teniem fent crides a les funcions de percolació per tots els grafs generats i guardant-nos els resultats de si complien o no les propietats per fer els gràfics.

2.3.3 Binomial Random Graphs

```
def binomial_graph_percolation(percolation_func, x_label, directory):
      """
      Generates plots given a percolation function about their
      connectivity and complexity

      :param percolation_func: Percolation function that will be used
      :param x_label: Label located in the x-axis of the plot
      :param directory: Directory where the plot will be saved
      :return: None
      """
      # We try for every probability 10 times Ex: if two times the graph
        is connected then we have a 20%
      times = 10
      # probability that it is indeed connected
      nplot = 0
      node_values = [10, 20, 50, 100, 500, 1000, 2000, 5000, 10000]
      p_gen_connected_graph = [0.5, 0.30000000000000004, 0.2, 0.1, 0.1,
                               0.1, 0.1, 0.1, 0.1
                              ]
      p_gen = 0
      for Nnodes in tqdm(node_values, desc="Nodes"):
          numbers_x = []
          numbers_y = []
          numbers_y_complex = []
          numbers_y_complex_and_connected = []
          chosen_p_q = p_gen_connected_graph[p_gen]
          for probQ in tqdm(np.linspace(0, 1, 11), desc="Probability",
                            leave=False
                           ):
            n_connected = 0
            n_complex = 0
            n_complex_and_connected = 0
            bi_graph = read_graph("/binomial_graph/graphs/", Nnodes,
                                  chosen_p_q, 0,
                                  ReadGraphOption.binomial
                                 )
```

```
        for _ in tqdm(range(times), desc="Time", leave=False):
            n_complex, n_complex_and_connected, n_connected =
            percolate_graph_info(bi_graph, n_complex,
                                 n_complex_and_connected,
                                 n_connected, percolation_func,
                                 probQ
                                )




        calculate_prob_connex_complex(n_complex,
                                      n_complex_and_connected,
                                      n_connected, numbers_x,
                                      numbers_y, numbers_y_complex,
                                     numbers_y_complex_and_connected,
                                      probQ, times
                                      )
    # plot graph connected
    connected_plot(numbers_x, numbers_y, x_label, nplot, Nnodes,
                   directory
                  )
    # plot graph complex
    complex_plot(numbers_x, numbers_y_complex, x_label, nplot, Nnodes,
                 directory
                )
    # plot graph complex and connected
    complex_and_connected_plot(numbers_x,
                               numbers_y_complex_and_connected,
                               x_label, nplot, Nnodes, directory
                              )
    nplot += 1
    p_gen = p_gen + 1
    reset_plots()
```

Per percolar els grafs Binomials hem implementat la funció binomial_graph_percolation. En aquesta funció percolem cada graf 10 vegades, percolem el graf que el llegim d'un fitxer .txt que hem generat prèviament. Més concretament el que fem és: usant l'estudi previ sobre la connexitat dels grafs aleatoris binomials, triem una probabilitat que podem assegurar que el graf generat és connex (vegeu en experiments l'apartat: 3.2.1 Estudi de transició de fase connexitat). Per a cada node n i per a la probabilitat que tenim a la llista `p_gen_connected_graph`, anem llegint els grafs(funcion `read_graph`) que estan guardats en fitxers .txt(mitjançant llistes d'adjacència) que tinguin la probabilitat >= a

la de llista esmentada anteriorment per tant això ens assegura amb bastanta seguretat que el graf escollit és connex i aquest el percolem 10 vegades.
La informació que ens interessa és la següent: que els grafs percolats siguin connexos (variable n_connected), que sigui complex i que no sigui connex, que el graf sigui complex (variable n_complex) i que el graf sigui connex i complex alhora (variable n_ complex_and connected).
El procés de percolació el realitzem amb probabilitats que van de 0 a 1.0 en increments de 0.1, utilitzant `percolate_graph_info` obtenim la informació esmentada prèviament sobre les propietats dels grafs. I la funció següent ens calcula amb quina probabilitat després de percolar els grafs, els grafs resultants són connexos `calculate_prob_connex_complex`.

Guardem a llistes(numbers_) els resultats i els representem mitjançant gràfiques utilitzant les funcions _plot.

2.3.4 Random Geometric Graphs

```
def random_geometric_graph_percolation(percolation_func, x_label,
                                       directory
                                      ):
    """
    Generates plots given a percolation function about their
    connectivity and complexity

    :param percolation_func: Percolation function that will be used
    :param x_label: Label located in the x-axis of the plot
    :param directory: Directory where the plot will be saved
    :return: None
    """
    if not os.path.isdir(directory_path + directory):
    os.makedirs(directory_path + directory)
    # We try for every probability 10 times Ex: if two times the graph
      is connected then we have a 20%
    times = 10
    # probability that it is indeed connected
    nplot = 0
    node_values = [10, 20, 50, 100, 500, 1000, 2000, 5000, 10000]
    r_gen_connected_graph = [0.565685424949238, 0.565685424949238,
                             0.565685424949238, 0.282842712474619,
                             0.282842712474619, 0.282842712474619,
                             0.282842712474619, 0.282842712474619,
                             0.282842712474619
                            ]
    r_gen = 0
    for Nnodes in tqdm(node_values, desc="Nodes"):
        numbers_x = []
        numbers_y = []
        numbers_y_complex = []
        numbers_y_complex_and_connected = []
        chosen_r_q = r_gen_connected_graph[r_gen]
        for probQ in tqdm(np.linspace(0, 1, 11), desc="Probability",
                          leave=False
                         ):
          n_connected = 0
          n_complex = 0
          n_complex_and_connected = 0
          # n_connected_edge = 0
          geo_graph = read_graph("/random_geometric_graph/graphs/",
                                 Nnodes, chosen_r_q, 0,
                                 ReadGraphOption.geometric
                                )
```

```
        for _ in tqdm(range(times), desc="Time", leave=False):
            n_complex, n_complex_and_connected, n_connected =
            percolate_graph_info(geo_graph, n_complex,
                                 n_complex_and_connected,
                                 n_connected, percolation_func,
                                 probQ
                                )

        calculate_prob_connex_complex(n_complex,
                                      n_complex_and_connected,
                                      n_connected, numbers_x,
                                      numbers_y, numbers_y_complex,
                                    numbers_y_complex_and_connected,
                                      probQ, times
                                     )
    # plot graph connected
    connected_plot(numbers_x, numbers_y, x_label, nplot, Nnodes,
                   directory
                  )
    # plot graph complex
    complex_plot(numbers_x, numbers_y_complex, x_label, nplot, Nnodes,
                 directory
                )
    # plot graph complex and connected
    complex_and_connected_plot(numbers_x,
                               numbers_y_complex_and_connected,
                               x_label, nplot, Nnodes, directory
                              )
    nplot += 1
    r_gen = r_gen + 1
    reset_plots()
```

Aquesta funció s'encarrega de generar gràfiques que il·lustren com varia el fet que un graf generat pel mètode random geometric sigui connex, o complex, o complex i connex després d'aplicar una funció de percolació.

Per a escollir els grafs hem utilitzat valors de generació del radi a partir dels quals sabem amb prou seguretat que el graf serà connex sempre, aquests grafs han sigut generats prèviament per una altra funció i només ens cal llegir-los del disc. El funcionament d'aquesta funció és anàleg al de la 2.3.3 les diferències les veiem en els valors r_gen_connected que ens asseguren que el graf escollit serà connex i en què ara fem servir els grafs random geometric.

2.3.5 Graella

```
def percolate_graella(percolation_func, x_label, directory):
      """
      Generates plots given a percolation function about their
      connectivity and complexity

      :param percolation_func: Percolation function that will be used
      :param x_label: Label located in the x-axis of the plot
      :param directory: Directory where the plot will be saved
      :return: None
      """
      if not os.path.isdir(directory_path + directory):
      os.makedirs(directory_path + directory)
      # We try for every probability 10 times Ex: if two times the graph
        is connected then we have a 20%
      times = 10
      # probability that it is indeed connected
      nplot = 0
      nxn_values = [4, 7, 10, 23, 32, 45, 71, 100]
      for Nnodes in tqdm(nxn_values, desc="Nodes"):
          numbers_x = []
          numbers_y = []
          numbers_y_complex = []
          numbers_y_complex_and_connected = []
          for probQ in tqdm(np.linspace(0, 1, 11), desc="Probability",
                            leave=False
                           ):
            n_connected = 0
            n_complex = 0
            n_complex_and_connected = 0
            graella_to_percolate = read_graph("/graella/graphs/",
                                              Nnodes*Nnodes, probQ, 0,
                                              ReadGraphOption.graella
                                             )
            for _ in tqdm(range(times), desc="Time", leave=False):
                  n_complex, n_complex_and_connected, n_connected =
                  percolate_graph_info(graella_to_percolate, n_complex,
                                       n_complex_and_connected,
                                       n_connected, percolation_func,
                                       probQ
                                      )
```

```
        calculate_prob_connex_complex(n_complex,
                                      n_complex_and_connected,
                                      n_connected, numbers_x,
                                      numbers_y, numbers_y_complex,
                                    numbers_y_complex_and_connected,
                                      probQ, times
                                     )
    # plot graph connected
    connected_plot(numbers_x, numbers_y, x_label, nplot,
                   Nnodes* Nnodes, directory
                  )
    # plot graph complex
    complex_plot(numbers_x, numbers_y_complex, x_label, nplot,
                 Nnodes*Nnodes, directory
                )
    # plot graph complex and connected
    complex_and_connected_plot(numbers_x,
                               numbers_y_complex_and_connected,
                               x_label, nplot, Nnodes * Nnodes,
                               directory
                              )
    nplot += 1
    reset_plots()
```

A diferència de les anteriors percolacions en aquest tipus de graf no ens cal escollir una probabilitat a partir de la qual el graf és connex, ja que aquest graf és inherentment connex. A part d'això la funció fa el mateix que les anteriors, llegeix un graf del disc, el percola 10 vegades, obté els resultats i els afegeix al graf.

3. Experiments

A l'hora de fer els experiments, és a dir, comprovar si existeix una transició de fase per les propietats de si un graf en connex i de si les seves components connexes són complexes, havíem d'aplicar el procés de percolació sobre grafs connexos. Ja que si no ho eren la primera propietat ja no es podria donar per exemple.

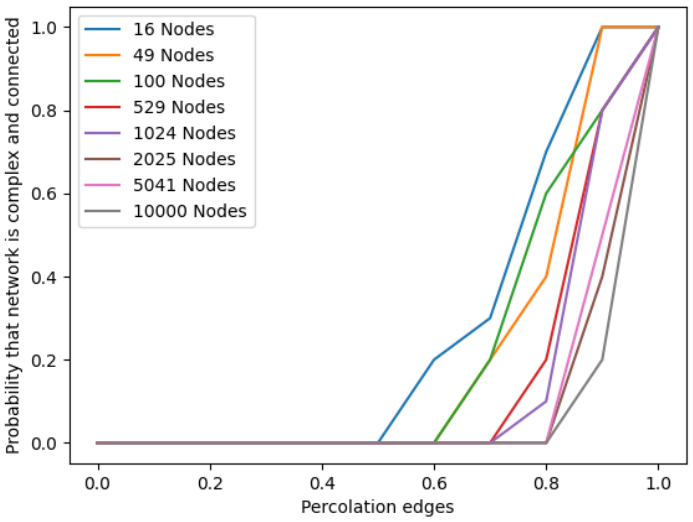
En el cas del graf graella aquesta condició ja es compleix, per tant, havíem d'estudiar per a un valor fixat de n que es donessin o no les 2 propietats depenent del paràmetre q del procés de percolació, tant per nodes com per arestes. Després anar incrementant n per observar la tendència i veure la transició de fase.

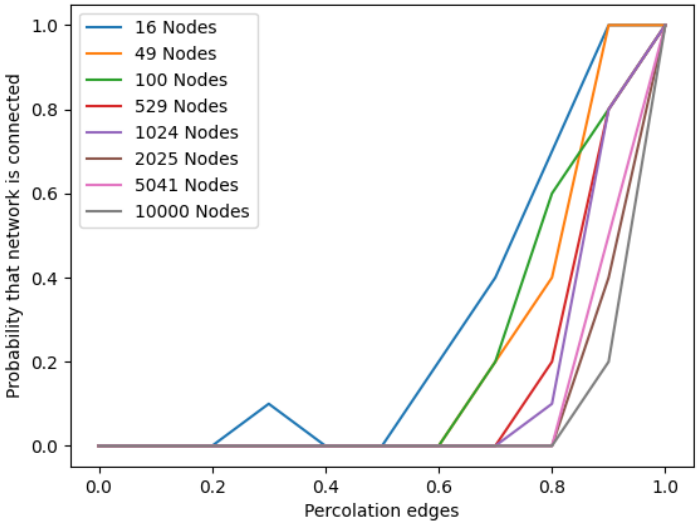
En el cas del binomial random graph G(n, p) havíem de fer primer un estudi per veure amb quins valors de p podíem assegurar que el graf fos connex. El que havíem de trobar era bàsicament que hi hagués una transició de fase a $p\pi$, de tal manera que per a $p\pi < p$ el graf G(n, p) fos amb alta probabilitat connex. Després sobre aquests grafs connexos fer igual que amb el graf graella, estudiar les 2 propietats depenent del paràmetre q del procés de percolació, tant per nodes com per arestes, i anar incrementant n per observar la tendència i veure la transició de fase.

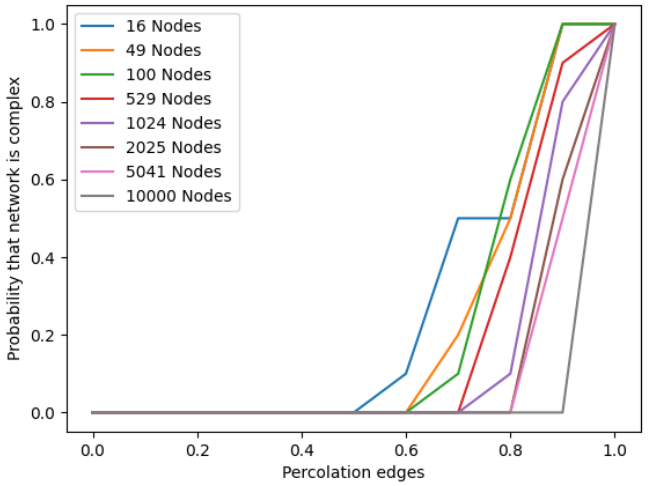
En el cas del random geometric graph G(n, r), igual que el model anterior, primer havíem de fer un estudi per veure amb quins valors de r podíem assegurar que el graf fos connex. Igual que abans havíem de trobar que hi hagués una transició de fase $r\pi$, de tal manera que per a $r\pi < r$ el graf G(n, r) fos amb alta probabilitat connex. Després sobre aquests grafs connexos fer el mateix que amb els models anteriors, estudiar les 2 propietats depenent del paràmetre q del procés de percolació, tant per nodes com per arestes i anar incrementant n per veure la tendència i veure la transició de fase.

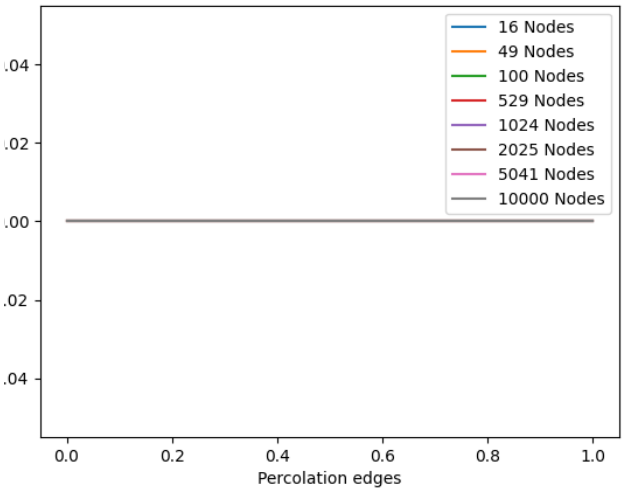
3.1 Graelles

3.1.1 Percolació per Arestes





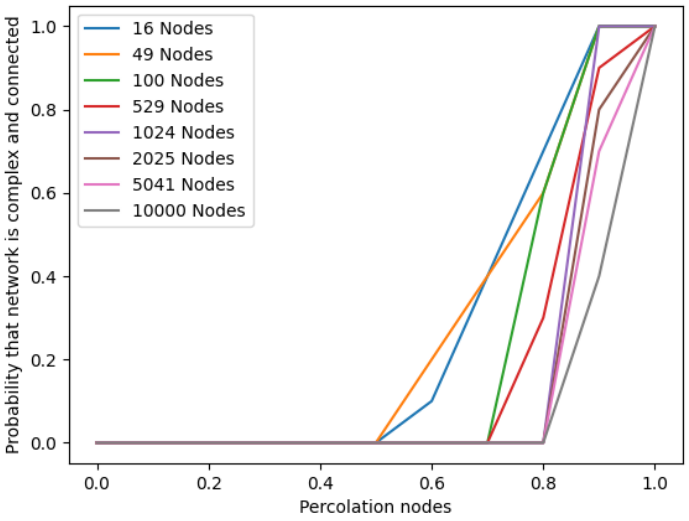


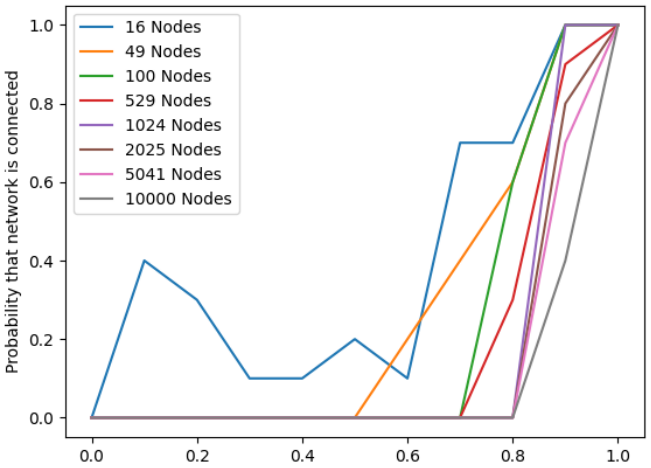


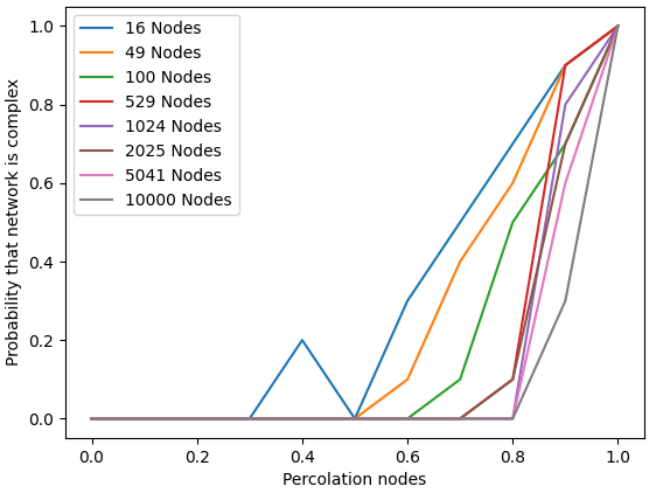
Gràfic complex i no connex

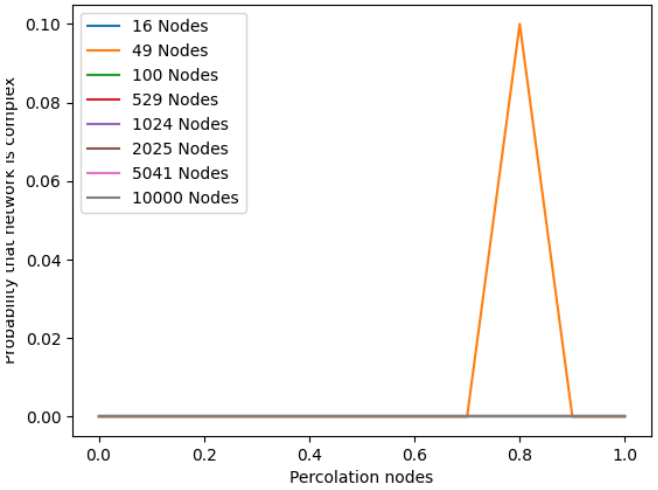
Pel procés de percolació per arestes del graf graella hem generat grafs amb *n* x *n* vèrtexs i per cada un d'aquests hem aplicat 10 vegades el procés de percolació per q ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut els gràfics anteriors, en els que podem veure que fins a *q* entre 0.9 i 1 no podem assegurar amb alta probabilitat que el graf percolat sigui connex.

3.1.2 Percolació per Nodes







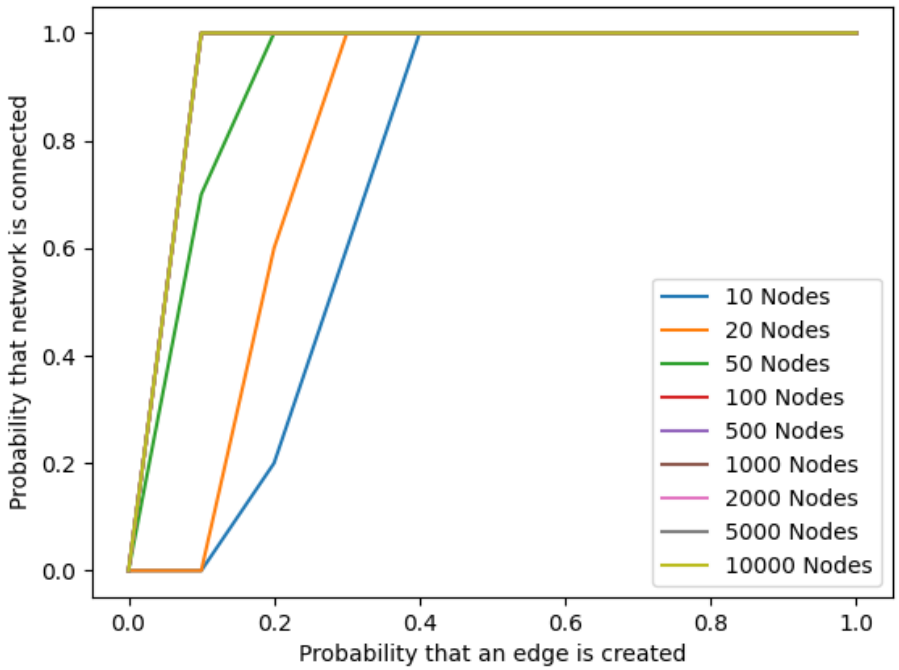


Gràfic complex i no connex

Pel procés de percolació per nodes del graf graella hem generat grafs amb *n* x *n* vèrtexs i per cada un d'aquests hem aplicat 10 vegades el procés de percolació per q ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut els gràfics anteriors, en els que podem veure que fins a *q* entre 0.9 i 1 no podem assegurar amb alta probabilitat que el graf percolat sigui connex.

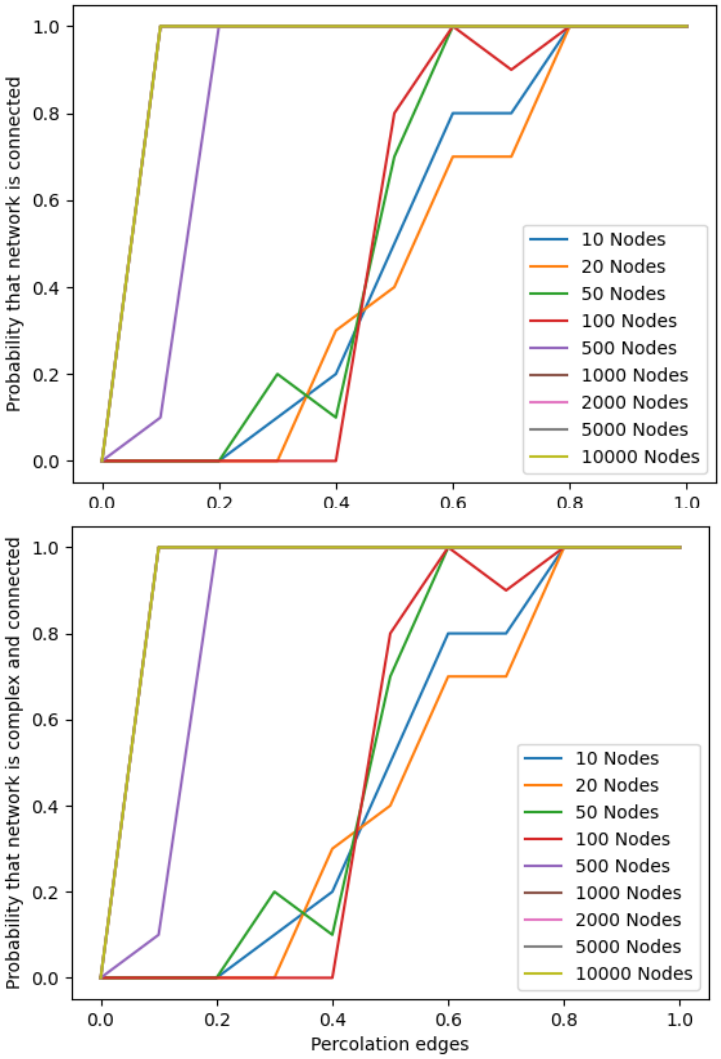
3.2 Binomial Random Graphs

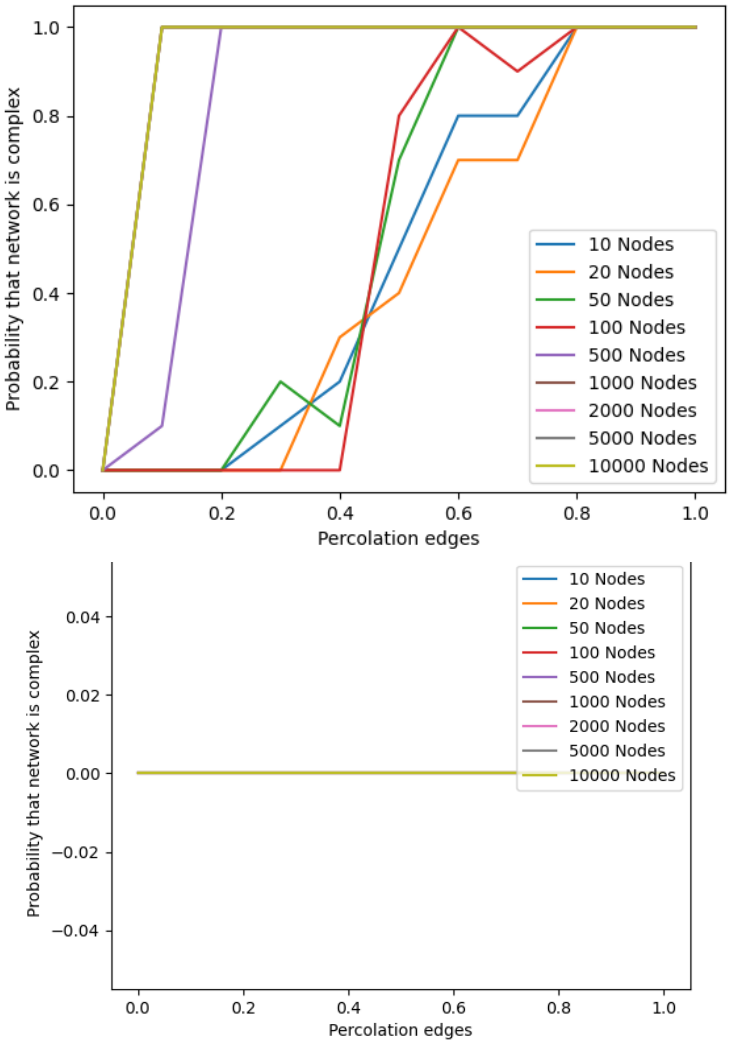
3.2.1 Estudi transició de fase connexitat



Per saber per quina p podíem assegurar que els binomial random graph G(n, p) eren connexos hem generat 10 grafs per cada n ∈ (5, 10, 20, 40, 50, 60, 80, 100, 500, 1000, 2000, 5000, 10000) i per cada per p ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut el gràfic anterior, en el que podem veure que hi ha una transició de fase per a p = 0.4, per a p > 0.4 podem assegurar que el graf generat és connex. A l'hora de fer la percolació tractarem cada nombre de nodes diferent amb la seva *p*.

3.2.2 Percolació per arestes

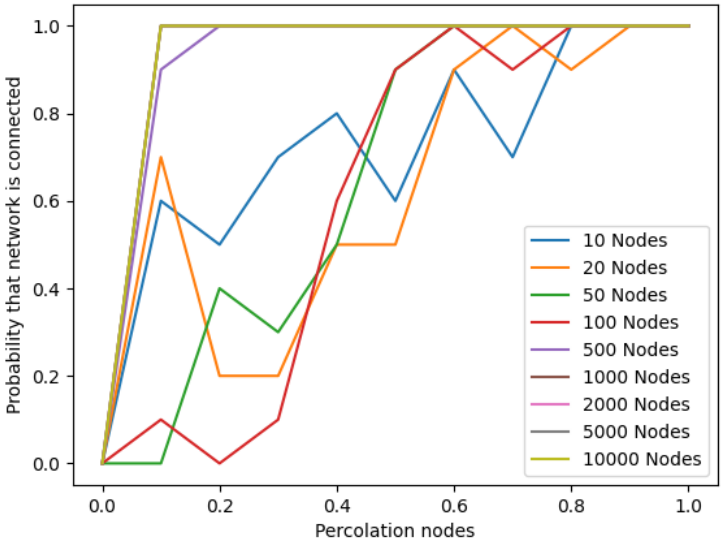


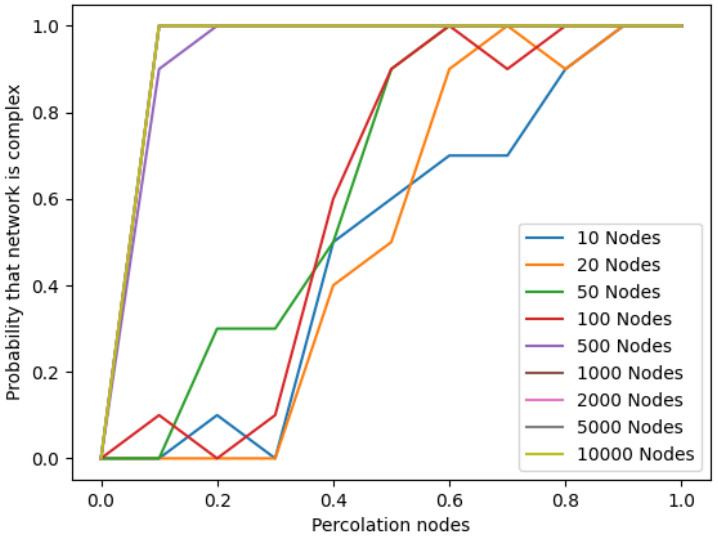


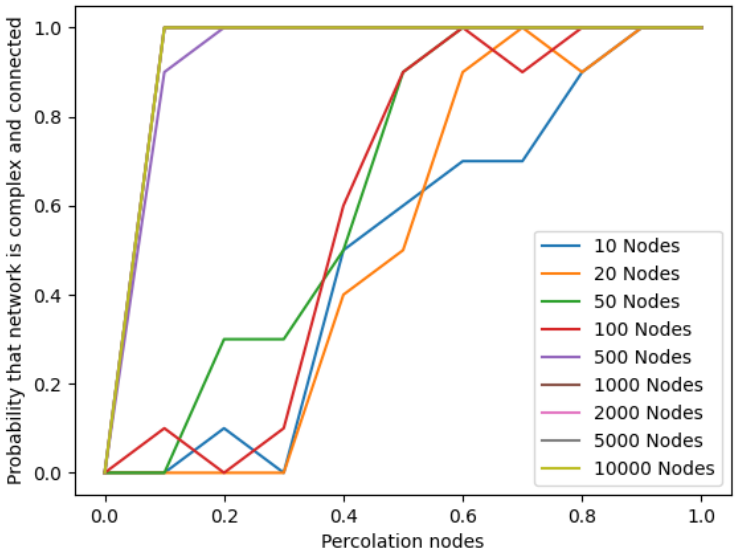
Gràfic complex i no connex

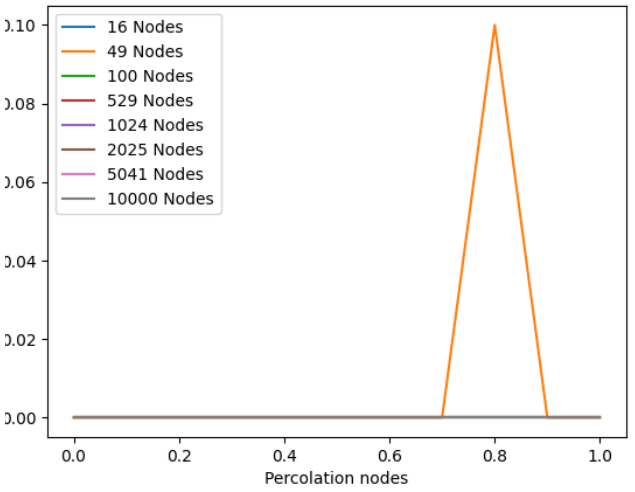
Pel procés de percolació per arestes del binomial random graph hem generat un graf per a n ∈ (10, 20, 50, 100, 500, 1000, 2000, 5000, 10000) i per a cada n la seva corresponent p que ens assegurava que el graf era connex, p ∈ (0.5, 0.3, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1), de tal manera que generàvem el G(10, 0.5), G(20, 0.3), G(50, 0.2), …. Per cada un d'aquests hem aplicat 10 vegades el procés de percolació per q ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut els gràfics anteriors, en els que podem veure que fins a *q* no podem assegurar amb alta probabilitat que el graf percolat sigui connex.

3.2.3 Percolació per nodes







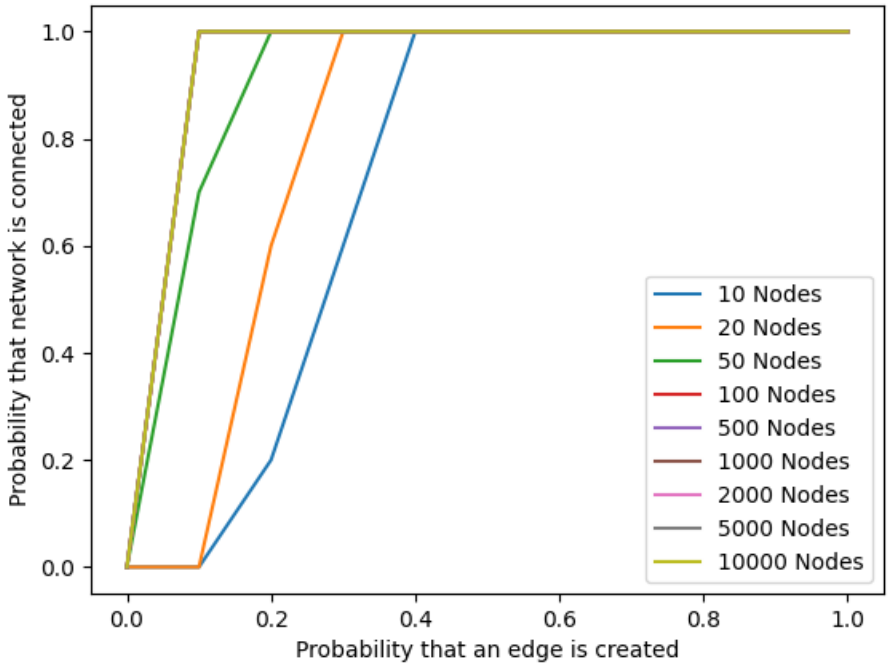


Gràfic complex i no connex

Pel procés de percolació per nodes del binomial random graph hem generat un graf per a n ∈ (10, 20, 50, 100, 500, 1000, 2000, 5000, 10000) i per a cada n la seva corresponent p que ens assegurava que el graf era connex, p ∈ (0.5, 0.3, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1), de tal manera que generàvem el G(10, 0.5), G(20, 0.3), G(50, 0.2), …. Per cada un d'aquests hem aplicat 10 vegades el procés de percolació per q ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut els gràfics anteriors, en els que podem veure que fins a *q* no podem assegurar amb alta probabilitat que el graf percolat sigui connex.

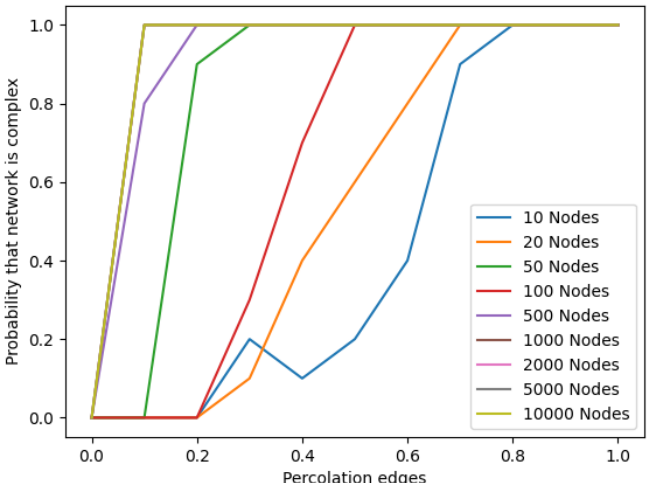
3.3 Random Geometric Graphs

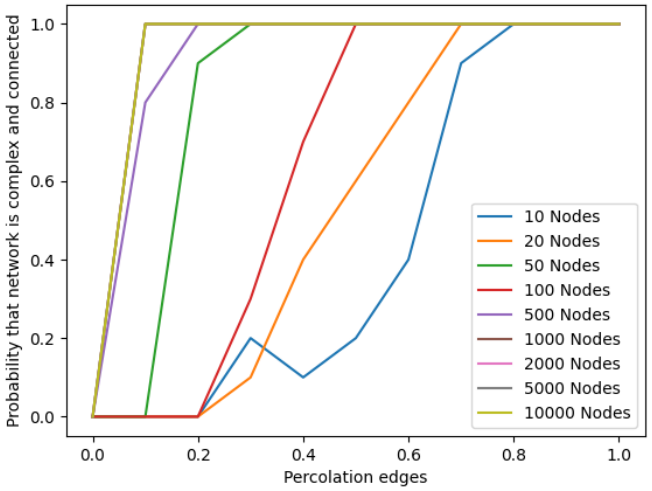
3.3.1 Estudi transició de fase connexitat

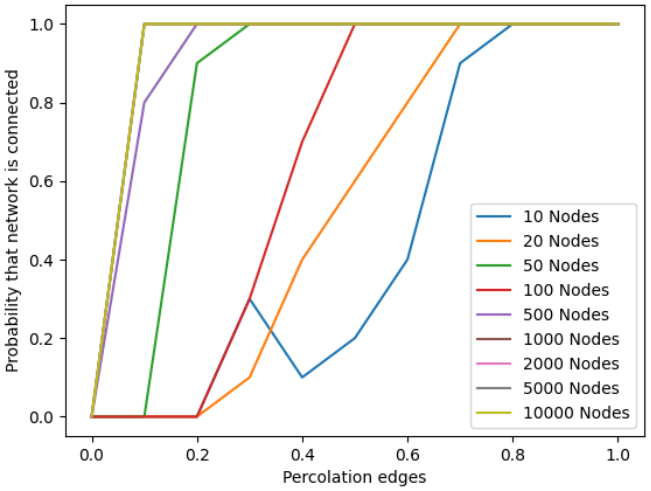


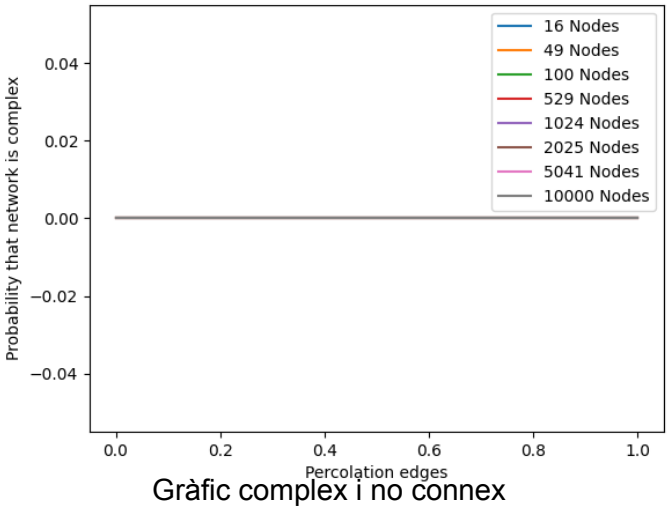
Per saber per quina r podíem assegurar que els random geometric graph G(n, r) eren connexos hem generat 10 grafs per cada n ∈ (5, 10, 20, 40, 50, 60, 80, 100, 500, 1000, 2000, 5000, 10000) i per cada per r ∈ linspace(0, 1, 11), que seria (0.1, 0.2, 0.3, 0.4,...). Fent els percentatges hem obtingut el gràfic anterior, en el que podem veure que hi ha una transició de fase per a p = 0.4, per a p > 0.4 podem assegurar que el graf generat és connex. A l'hora de fer la percolació tractarem cada nombre de nodes diferent amb la seva *r*.

3.3.2 Percolació per arestes









Gràfic complex i no connex

Tenim 4 gràfics:

En el gràfic del complex podem observar que quan augmentem el nombre de nodes la probabilitat de què després de fer la percolació el graf segueixi sent complex també augmenta.

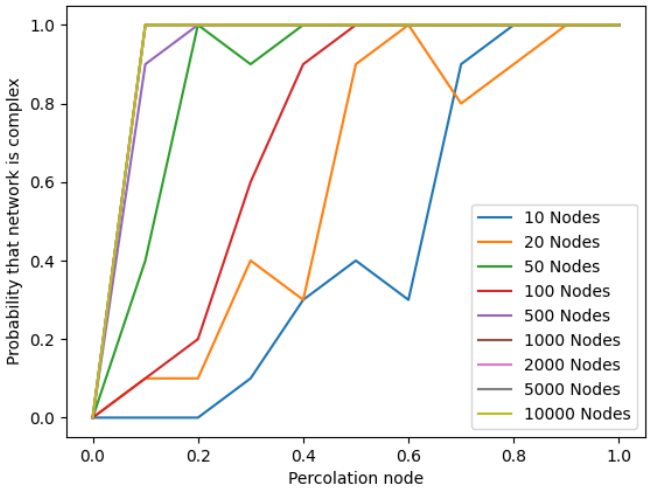
En el gràfic complex i connex podem veure que les propietats es conserven de forma similar al graf mencionat prèviament.

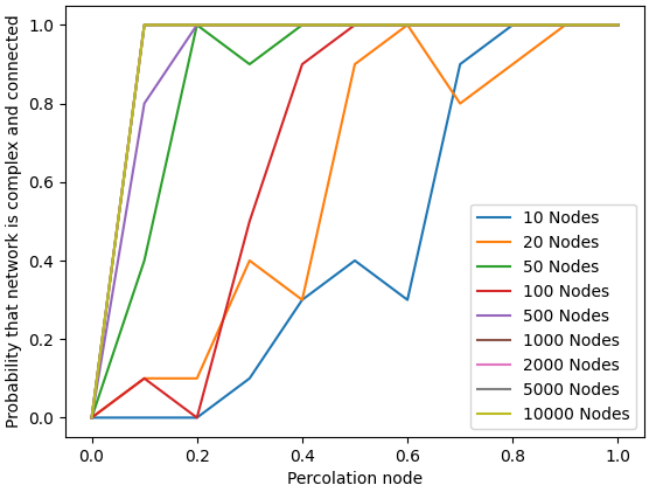
En el gràfic del connex veiem que segueix la tendència que en augmentar el nombre de nodes la propietat es manté amb major probabilitat després de la percolació

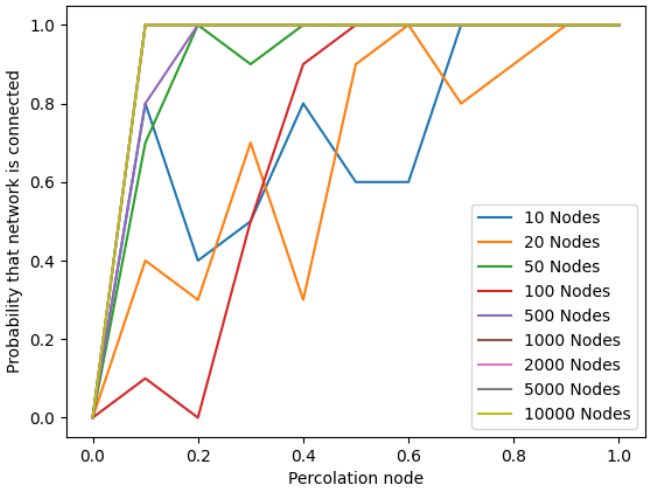
En el gràfic del complex i no connex veiem un cas peculiar en què tot és zero, això és degut al fet que hem sacrificat una mica de la precisió pel temps. Abans

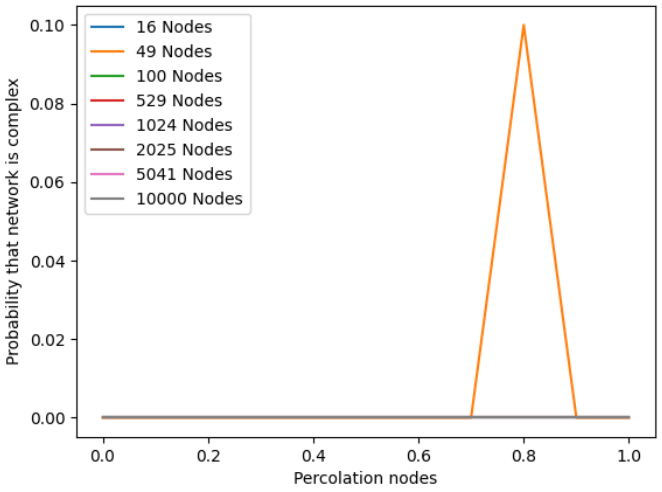
generàvem els grafs de forma dinàmica sense guardar-los en fitxers .txt, per tant, podíem tenir més precisió i l'execució es realitzava en un temps més ràpid. Però en guardar els fitxers al sistema i llegir-los i tornar-los a percolar perdem eficiència, també degut al gran nombre de nodes amb què tractem. Per aquesta raó hem reduït el nombre d'iteracions en percolar un graf. Ara realitzem 10 iteracions de percolació per graf. Abans en fèiem 50.

3.3.3 Percolació per nodes









Gràfic complex i no connex

Tenim 4 gràfics:

En el gràfic del complex podem observar que la tendència general és que quan augmentem el nombre de nodes la probabilitat de què després de fer la percolació el graf segueixi sent complex també augmenta, a més a més podem observar que hi ha més variabilitat, creiem que això és degut al fet que un graf pot esdevenir connex arran que en extreure un node que no és connex el graf pot esdevenir connex, propietat que no és possible en la percolació per arestes.

En el gràfic complex i connex podem veure que les propietats es conserven de forma similar al graf mencionat prèviament.

En el gràfic del connex veiem que segueix la tendència que en augmentar el nombre de nodes la propietat es manté amb major probabilitat després de la percolació però aquesta vegada amb encara més variabilitat que en els grafs complexos.

En el gràfic del complex i no connex veiem un cas peculiar per a 49 nodes, en els altres nodes passa el mateix que en la percolació per arestes. Això també és degut al fet que la pressió és baixa, perquè compensem la precisió pel temps d'execució i per tenir més nodes en els nostres grafs.

4. Conclusions

L'objectiu del projecte era analitzar experimentalment l'existència o no de transició de fase en processos de percolació de grafs, respecte a dues propietats, la de què un graf fos connex, que ja coneixíem, i la de què totes les components d'un graf fossin complexes, que hem après que era quan en una component connexa hi havia més d'un cicle.

Pel que fa a una transició de fase, coneixíem el concepte i l'havíem vist abans però no experimentat d'aquesta manera.

El procés de percolació també era una cosa nova per nosaltres de la qual no sabíem res, hem après i entès en què consistia i que és un fenomen que es pot donar en la realitat, que falli el que en la representació de la realitat per un graf correspon a un node o una aresta i que en aquests casos fer un estudi així pot ser molt útil.

El que també hem descobert amb el projecte són tres tipus nous de grafs: el graella, el binomial i el random geometric; els quals ara coneixem amb profunditat gràcies a l'estudi fet. Pel graf binomial i pel random geometric vam haver de fer un estudi previ perquè no qualsevol graf d'aquest tipus és connex, cosa que havien de satisfer els grafs del projecte abans d'aplicar-los el procés de percolació, hem arribat a la conclusió, que ja suposàvem des d'un bon principi que com més gran sigui la p en el cas del graf binomial i la r en el cas del graf random geometric, més probabilitats té el graf de ser connex, perquè més arestes tindrà, gràcies a l'estudi hem trobat les p i les r que són la transició de fase per la propietat de connectivitat per aquests grafs, les quals estan al voltant de 0,4. Els grafs graella són sempre connexos , per tant, ja complien els requisits i no calia fer cap estudi previ.

Hem fet els experiments amb un nombre força alt de grafs i de nodes nosaltres creiem i hem obtingut el resultat que esperàvem, tot i que per alguns potser faria falta una mica més de precisió.

Una de les conclusions que arribem també és que com major és el nombre de nodes dels grafs amb què hem experimentat més probabilitat tenen de ser connexos.

5. Bibliografia

[1] Wikipedia, Erdős–Rényi model.
https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model

[2] Wikipedia, Random geometric graph.
https://en.wikipedia.org/wiki/Random_geometric_graph

[3] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008
https://networkx.org/documentation/stable/reference/algorithms/index.html

[4] R. Sedgewick and K. Wayne. Algorithms. Addison-Wesley, 4th edition, 2011.
https: //algs4.cs.princeton.edu/

[5] I. E. Antoniou, E. T. Tsompa, "Statistical Analysis of Weighted Networks", *Discrete Dynamics in Nature and Society*, vol. 2008, Article ID 375452, 16 pages, 2008.
https://doi.org/10.1155/2008/375452

[6] Jesper Dall, Michael Christensen, "Random Geometric Graphs", Physical Review E, American Physical Society, 2002.
https://arxiv.org/abs/cond-mat/0203026v2