VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Grafo šerdis

The core of a network

Kursinis darbas

Atliko: 3 kurso 3 grupės studentas
Miglė Šimavičiūtė (parašas)

Darbo vadovas: dr. Mindaugas Bloznelis (parašas)

Vilnius – 2023

TURINYS

ĮVADAS 3
1. Sąvokos 4
2. Dvidalio grafo modelis 6
3. Aktorių grafas 8
4. K – šerdies modelis 9
5. Grafo šerdies radimo pavyzdys 11
6. Išvados 16
6. Šaltiniai 17

ĮVADAS

Grafų teorija yra matematikos šaka, tirianti grafus bei ryšius tarp jų. Grafai susideda iš viršūnių bei briaunų aibių, kurios parodo jungumą tarp grafo viršūnių. Grafo modelis gali būti naudojamas įvairiose srityse, pradedant socialiniais tinklais ir baigiant biologinėmis sistemomis.

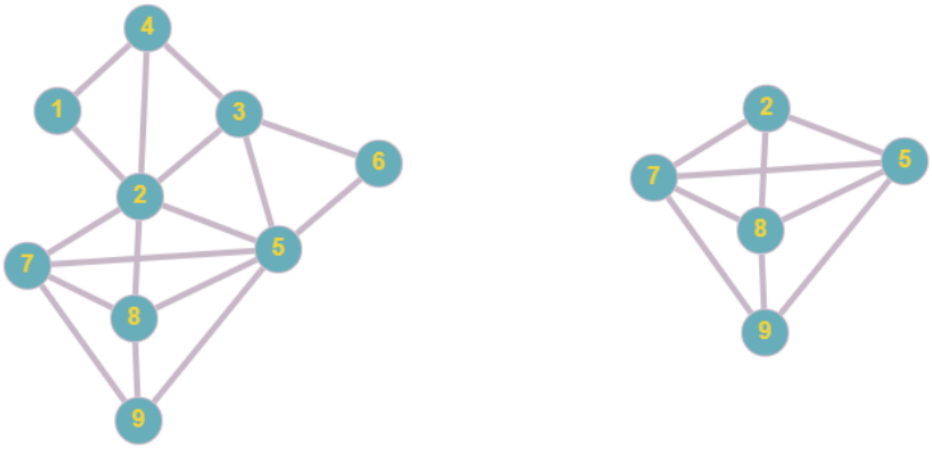
Šiame darbe nagrinėsiu dvidalio grafo, aktorių grafo bei k – šerdies sąvokas bei jų modelius, realizavimą. Realizavimui pasitelksiu Python programavimo kalbą, o visą teoriją pritaikysiu praktiškai.

1. Sąvokos

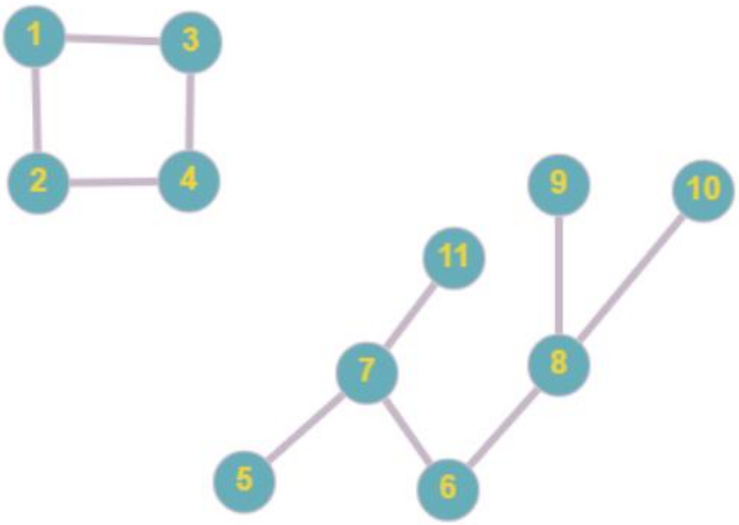
Grafas - matematikoje tai abstrakti struktūra, skirta vizualizuoti ryšius tarp skirtingų objektų. Grafas susideda iš viršūnių aibės V ir juos jungiančių briaunų aibės E . Žymėjimas:

$$G = (V, E)$$

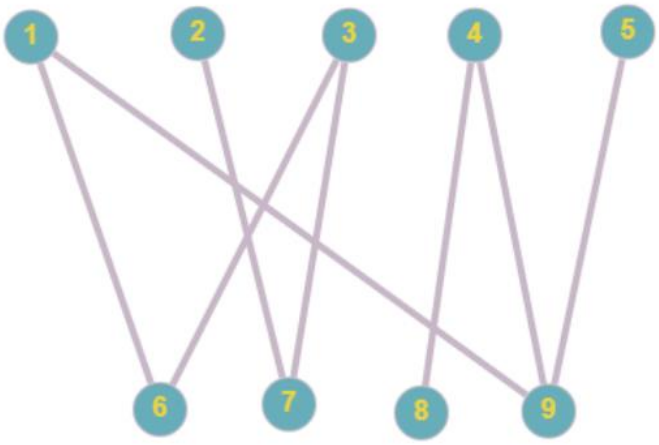
Grafo k-šerdis - tai grafo prografis, kurio kiekviena viršūnė turi k arba daugiau incidenčių briaunų. Taip atrodo grafo (kairėje) k-šerdis, kai k = 3 (dešinėje).



Grafo komponentė - tai grafo pografis, kuriame bet kurias dvi viršūnes jungia viena ar daugiau briaunų. Nuotraukoje esantis grafas turės dvi komponentes.



Dvidalis grafas – tai grafas, kurio viršūnių aibę galima išskirti į dvi atskiras viršūnių aibes, taip, kad kiekviena briauna jungtų tik viršūnes iš skirtingų aibių.
Grafą paveiksliuke galime išskirti į dvi viršūnių aibes: {1, 2, 3, 4, 5} ir {6, 7, 8, 9}.



Aktorių grafas – tai grafas, kurį gauname iš dvidalio grafo, kurio viena grupę viršūnių laikome aktoriais, o kitą grupę galime laikyti atributais. Aktorių grafas bus sudarytas iš aktorių viršūnių aibės ir viršūnių porą jungs briauną, jei abi viršūnės turės briauną su tuo pačiu atributu.

2. Dvidalio grafo modelis

Kad galėtume nagrinėti šerdį, mums reikalingas dvidalis grafas, tam pasinaudosiu algoritmu, kuriam sukurti reikalingi du masyvai n1 ir n2, kuriuose bus nurodyti elementų svoriai, taip pat bus reikalingas skaičius α = [0, 1], kuriuo pasinaudosiu skaičiuojant tikimybę, kuri nurodys ar būsimo dvidalio grafo viršūnės bus kaimynės ar ne.

Algoritmui paduodami trys parametrai: du viršūnių masyvai n1 ir n2 su svoriais x ir y bei α.

Algoritmas dvidalio grafo generavimui:

1. Inicializuojamas tuščias gretimų kaimynių sąrašas, kuriame saugosime dvidalio grafo viršūnes bei kaimynes. Į sąrašą pridedame viršūnes iš masyvų n1 ir n2.
2. Pradedamas ciklas, kiekvienos iteracijos metu iš eilės einama per masyvo n1 elementų svorius.
    a. Pradedamas dar vienas ciklas, kiekvienos iteracijos metu iš eilės einame per masyvo n2 elementų svorius.
        i. Apskaičiuoja viršūnių jungumo tikimybė
        $$p = \alpha \frac{(n1\ elemento\ svoris * n2\ sąrašo\ \ elemnto\ svoris)}{\sqrt{\boldsymbol{n1\ ilgis * n2\ ilgis}}}$$
        ii. Jei atsitiktinis skaičius intervale (0 ; 1) yra mažesnis nei p, tuomet į gretimų viršūnių sąrašą elementui iš n1 pridedama kaimynė iš n2, o elementui iš n2 pridedama kaimynė iš n1.
        iii. Ciklas kartojamas kol pereiname per visus n2 elementus.
    b. Ciklas kartojamas kol pereiname per visus n1 elementus.
3. Programa grąžina dvidalio grafo gretimų kaimynių sąrašą.

Kodas įgyvendintas naudojantis Python kalba, kodo paleidimui reikalingas modulis „Random“:

```
def random_bipartite_graph(n1, n2, alpha):
    V = len(n1) + len(n2)
    adj_list = [[] for i in range(V)]
    for i in range(len(n1)):
        f = 0
        for j in range(len(n1), V):
            p = getP(alpha, n1[i], n2[f], len(n1), len(n2))
```

```
            if random.random() < p:
                adj_list[i].append(j)
                adj_list[j].append(i)
            f += 1
    return adj_list
```

Turėdami sugeneruotą atsitiktinį dvidalį grafą, jį panaudosime aktorių grafo generavimui. Generuodami dvidalį grafą pagal viršūnių svorius bei α galėsime keisti tikimybę ar dvi viršūnės bus jungios ar ne, tai padės analizuoti grafo šerdį esant skirtingiems parametrams.

3. Aktorių grafas

Aktorių grafas yra svarbus, nes naudodamiesi jo modeliu galime analizuoti kaip bendri atributai jungia viršūnes. Aktorių grafo gavimui pasinaudosiu apačioje aprašytu algoritmu, kurį įgyvendinau naudojantis Python programavimo kalba, programai reikia paduoti aktorių skaičių bei grafo gretimų kaimynių sąrašą. [BD13] [BK16]

Algoritmas aktorių grafo radimui:

1. Sukuriame naują gretimų kaimynių sąrašą, kuriame saugosime aktorių grafą, į jį pridedame tiek viršūnių, kiek yra aktorių elementų.
2. Pradedamas ciklas, kurio metu einame per aktorių viršūnes.
    a. Pradedame dar vieną ciklą ir jo metu einame per duoto grafo gretimų kaimynių sąrašą.
        i. Jei dvi ne vienodos viršūnės turi vienodų kaimynių, į naujai sukurtą gretimų kaimynių šios viršūnės abiems viršūnės yra pridedamos kaip kaimynės.
    b. Ciklas kartojamas kol pereiname visą duotojo grafo gretimų kaimynių sąrašą.
3. Ciklas kartojamas kol pereiname visas aktorių viršūnes.
4. Programa grąžina aktorių grafo gretimų kaimynių sąrašą.

Kodas įgyvendintas naudojantis Python kalba:

```
def findConnectedActors(x, graph):
    neighbors = {}
    new_graph = {}
    for i in range(x):
        for neighbor in graph[i]:
            neighbors.setdefault(neighbor, []).append(i)
    for i in range(x):
        new_graph[i] = list({n for neighbor in graph[i] for n in
neighbors.get(neighbor, {}) if n != i})

    return new_graph
```

4. K-šerdies modelis

Turėdami aktorių grafą galime ieškoti jo k – šerdies. Jos radimą taip pat automatizavau, apačioje aprašiau algoritmą, jam reikalingi duomenys: natūralusis skaičius k bei aktorių grafo gretimų kaimynių sąrašas.

Algoritmas k-šerdies radimui aktorių grafe

1. Padaroma duoto grafo gretimų kaimynių sąrašo kopiją, ją naudosime šerdies saugojimui.
2. Pradedamas ciklas, einame per visas grafo viršūnes.
   a. Jei viršūnės laipsnis yra mažesnis už k:
      i. Viršūnė pašalinama iš grafo gretimų kaimynių sąrašo kopijos.
      ii. Viršūnė pašalinama iš kitų viršūnių kaimynių.
3. Ciklas kartojamas tol, kol yra viršūnių kurias galima pašalinti.
4. Programa grąžina atnaujintą gretimų kaimynių sąrašo kopiją, kuri ir bus grafo k – šerdis.

Kodas įgyvendintas naudojantis Python kalba:

```
def getCore(graph, k):
    core = graph.copy()
    degrees = getVertexDegrees(core)
    removed = True
    while removed:
        removed = False
        for i in range(len(graph)):
            if degrees[i] < k and degrees[i] > 0:
                for neighbor in core[i]:
                    core[neighbor].remove(i)
                    degrees[neighbor] -= 1
                core.pop(i)
                degrees[i] = 0
                removed = True
            elif degrees[i] == 0 and core.get(i) is not None:
                core.pop(i)
```

```
                removed = True
    return core
```

5. Grafo k – šerdies radimo pavyzdys

Turėdami įrankius dvidalio grafo, jo aktorių grafo bei k – šerdies radimui, galime juos pritaikyti praktiškai. Iš pradžių sugeneruosime atsitiktinį dvidalį grafą, jo radimui mums reikės aktorių bei atributų masyvų su svoriais, taip pat skaičiaus alpha, kuris nulems grafo viršūnių jungumą. [BK16]

Aktorių bei atributų svoriui masyvų generavimui pasinaudosime šia lygtimi:

$$X = \begin{cases} u^{-b}, & jei\ u^{-b} \geq A \\ A, & kitu\ atveju \end{cases}$$

Kintamasis u yra atsitiktinis skaičius iš intervalo (0;1), A ir b yra mūsų nuožiūra pateikiami skaičiai.
Šio pavyzdžio atveju abu masyvus sudarys 1000 elementų, A = 4, o aktorių svorių masyvo generavimui b = 3.5, o atributų svorių masyvo generavimui b = 0.2.

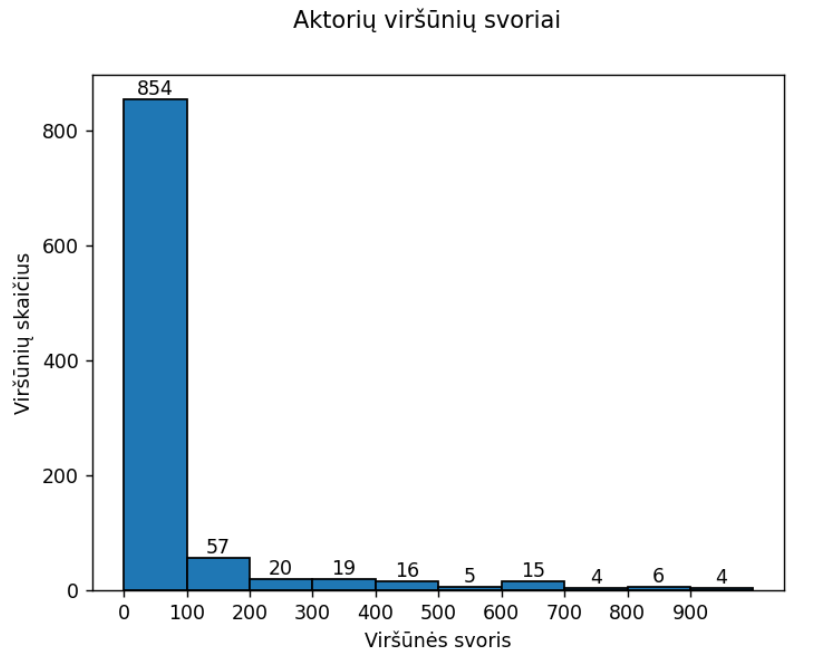
Lygtį realizavau Python progamavimo kalba:

```
def generateArray(A, a):
    array_length = 1000
    xArray = []
    for x in range(0, array_length):
        while True:
            u = round(pow(random.uniform(0.1, 1), -a))
            if (u >= A and u <= 1000):
                xArray.append(u)
                break
            if(u <= 1000):
                xArray.append(A)
                break
    return xArray
```

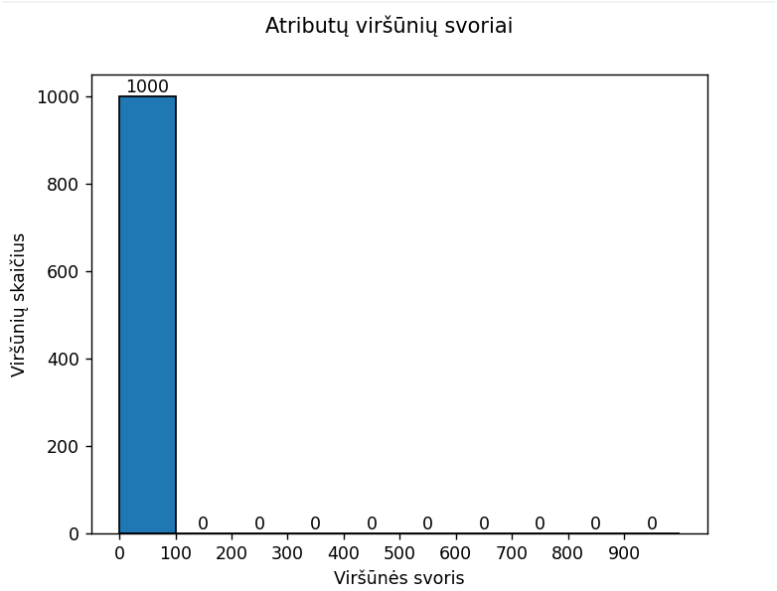
Taip pat svarbu aptarti grafo komponentes, nes tai yra vienas iš grafo bruožų, leidžiantis suprasti jo sandarą. Grafo komponenčių radimą realizavau Pythono programavimo kalba:

```
def find_components(graph):
    visited = set()
    components = []

    def dfs(node, component):
        visited.add(node)
        component.append(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs(neighbor, component)

    for node in graph:
        if node not in visited:
            component = []
            dfs(node, component)
            components.append(component)

    return components
```

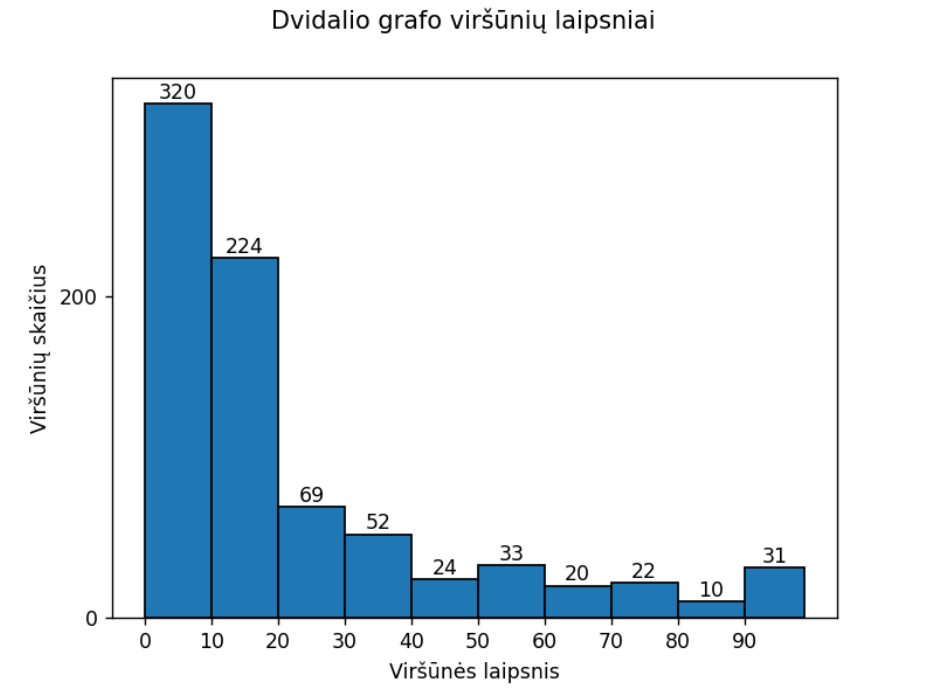
Pasinaudoję viršuje aprašytu kodu, gauname aktorių masyvą:



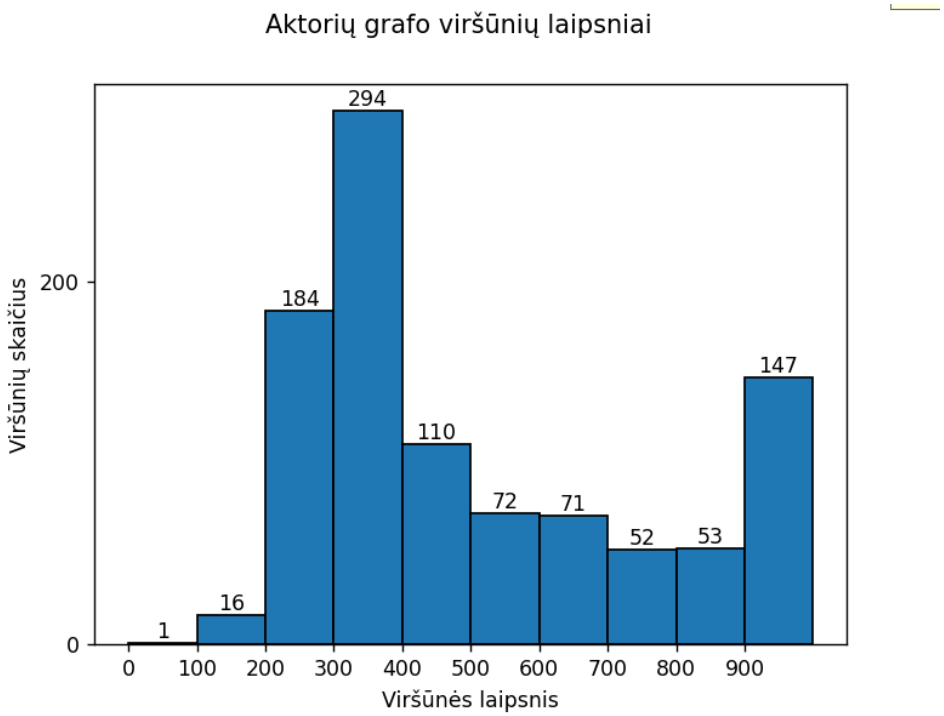
Bei atributų masyvą:



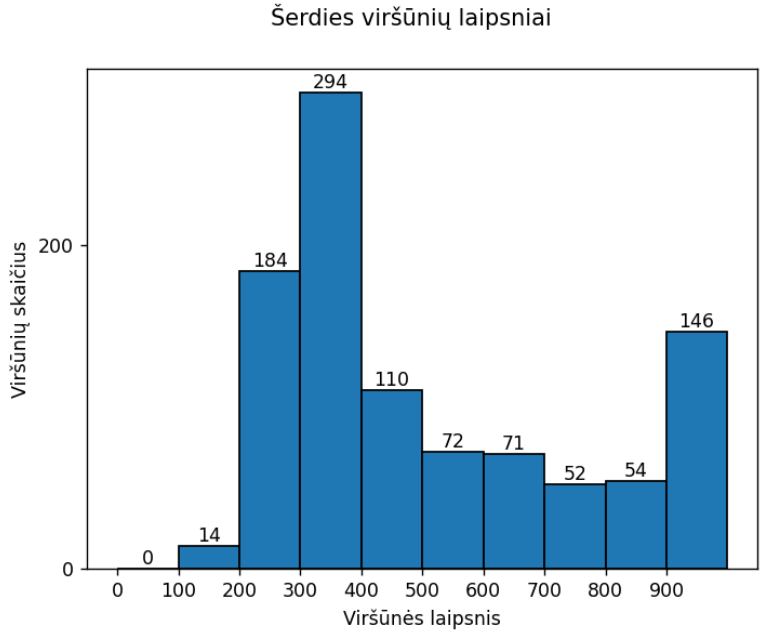
Turėdami aktorių bei atributų masyvus, galime iš jų sukurti dvidalį grafą, tam pasinaudosiu 2 skyriuje aprašytu algoritmu, gauname dvidalį grafą, kurio viršūnių laipsnių histograma atrodo taip ($\alpha = 0.5$):



Iš histogramos matyti, jog dauguma įvairų kieki kaimynių. Pasinaudojus trečiame skyriuje parašytu algoritmu gauname aktorių grafą, jo viršūnių laipsnių histograma atrodo taip:



Turėdami aktorių grafą galime rasti šerdį. Kadangi šis grafas turi didelį kiekį viršūnių, o jos tarpusavyje kaimynių, nusprendžiau šiam pavyzdžiui k reikšmę prilyginti 150. Pasinaudojus ketvirtame skyriuje aprašytu algoritmu, iš histogramos matyti, jog grafo šerdis, kai k = 150, netenka tik 3 viršūnių.



Taip pat galime peržvelgti ir kitas grafo savybes. Svarbi grafo dalis yra komponentės, šis grafas turės tik vieną komponentę ir ji bus sudaryta iš visų likusių 997 viršūnių.

6. Išvados

• Galima algoritmiškai aprašyti ir realizuoti programinį kodą dvidaliam grafui, aktorių grafui bei k – šerdies radimui.

• Nuo skirtingų grafo parametrų kinta ir jo k - šerdis.

ŠALTINIAI

[Mat10] Matuliauskas, K. "Grafų teorija." Vilnius University publish house (2010).

[BK16] Clustering function: Another view on clustering coefficient. Journal of Complex Networks.

[BD13] Degree distribution of an inhomogeneous random intersection graph. The Electronic Journal of Combinatorics.