به نام خدا

دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

تمرین دوم درس تحلیل شبکه های پیچیده

Community Detection

استاد درس: دکتر چهرقانی

نام: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

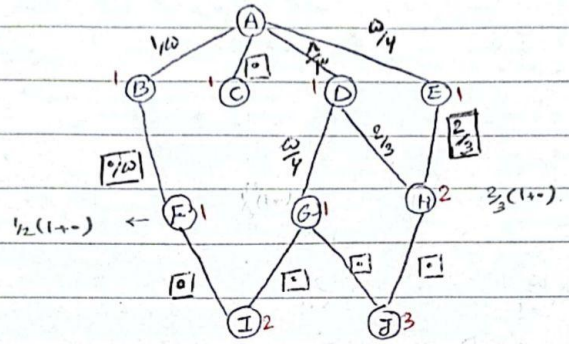
پاییز ۱۴۰۲

فهرست مطالب

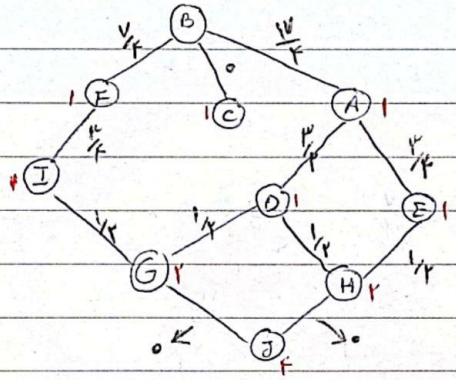
سوال اول

ابتدا باید ۱۰ گراف DAG به روش BFS از هر کدام از ۱۰ گره تشکیل بدهیم و مقدار betweenness centrality هر یال را در هر یک از ۱۰ گراف به دست بیاوریم. در نهایت میزان centrality یک یال برابر با جمع centrality های آن یال در این ۱۰ گراف است.

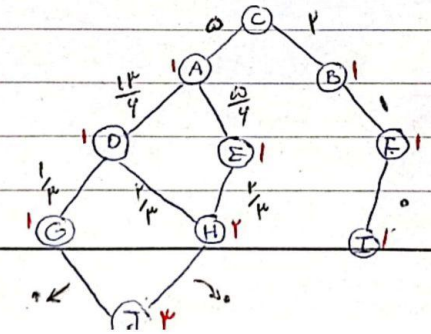
برای گره A:



برای گره B:

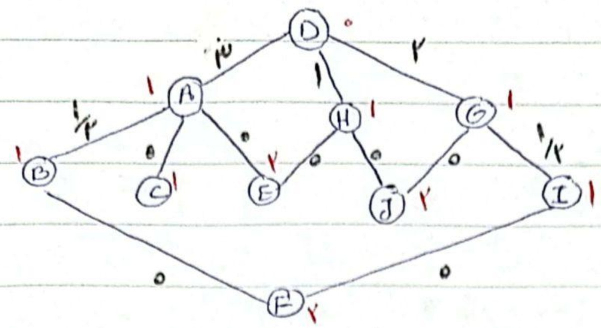


برای گره C:

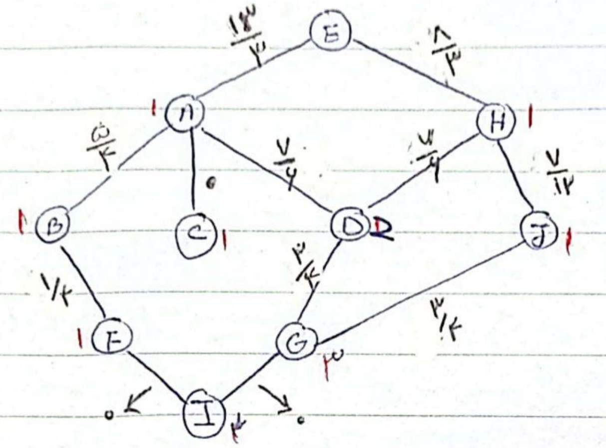


Senobar

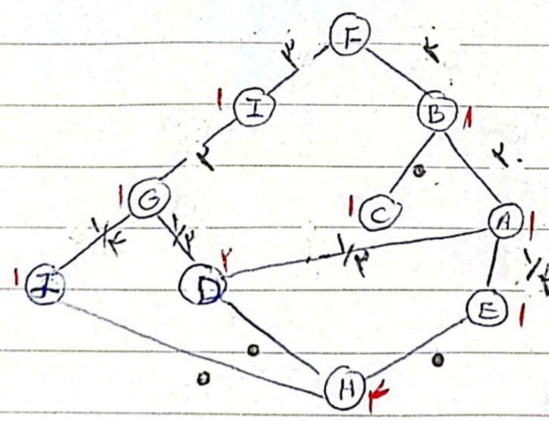
ریشه D :

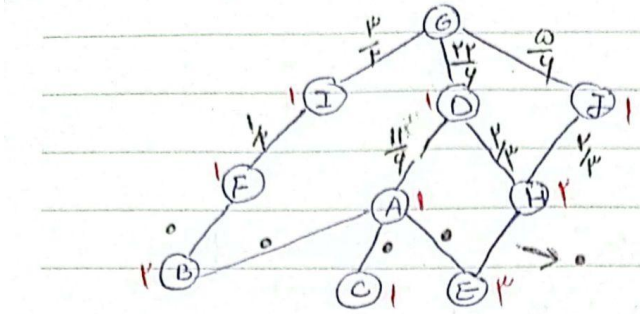


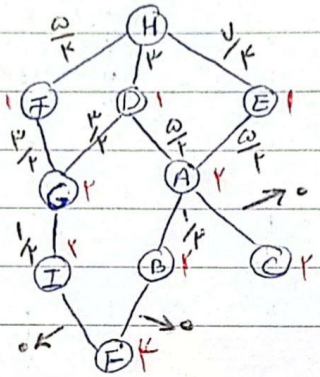
ریشه E :

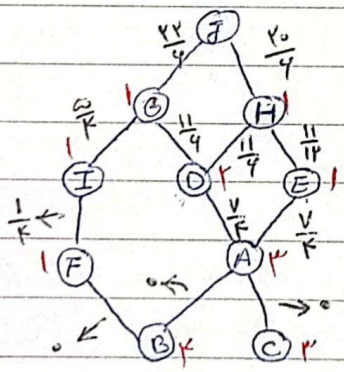


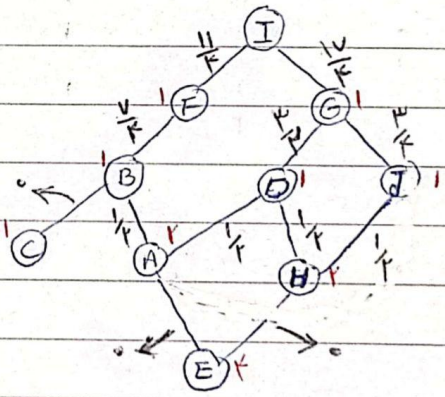
ریشه F :











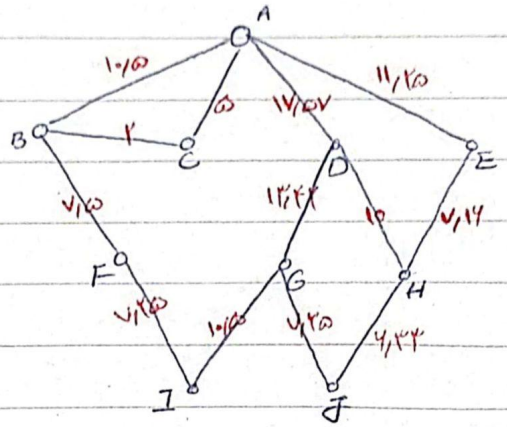
در جدول زیر Centrality هر یال در هر یک از گراف‌ها مشخص شده و برای بدست آمدن نتیجه نهایی این یال‌ها با هم جمع شده است.

ج)

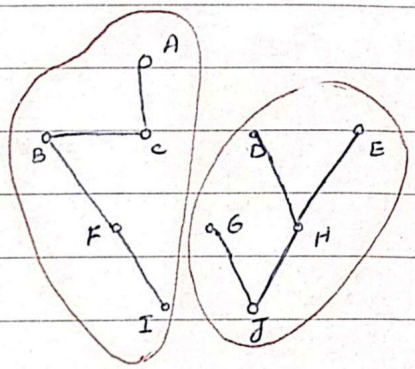
	A	B	C	D	E	F	G	H	I	J	Sum
AB	۱٫۵	۱۷/۴	۰	۱/۲	۵/۴	۲	۰	۱/۲	۱/۲	۰	۱۰٫۵
AC	۰	۰	۵	۰	۰	۰	۰	۰	۰	۰	۵
AD	۸/۳	۳/۲	۱۳/۶	۳	۷/۶	۱/۲	۱۱/۶	۵/۲	۱/۲	۷/۴	۱۷٫۵۷
AE	۵/۶	۳/۴	۵/۶	۰	۱۳/۳	۱/۴	۰	۵/۲	۰	۷/۴	۱۱٫۲۵
BC	۰	۰	۲	۰	۰	۰	۰	۰	۰	۰	۲
BF	۰٫۵	۷/۴	۱	۰	۱/۴	۴	۰	۰	۷/۴	۰	۷٫۵
DG	۵/۶	۱/۲	۱/۳	۲	۳/۴	۱/۲	۲۲/۶	۳/۲	۳/۲	۱۱/۶	۱۳٫۴۲
DH	۲/۳	۱/۲	۲/۳	۱	۷/۶	۰	۲/۳	۳	۱/۲	۱۱/۶	۱۰
EH	۲/۳	۱/۲	۲/۳	۰	۸/۳	۰	۰	۷/۴	۰	۱۱/۱۲	۷٫۱۶
HJ	۰	۰	۰	۰	۷/۱۲	۰	۲/۳	۵/۴	۱/۲	۲۰/۶	۶٫۴۳
GJ	۰	۰	۰	۰	۳/۴	۱/۴	۵/۶	۳/۲	۳/۴	۲۲/۶	۷٫۲۵
FI	۰	۳/۴	۰	۰	۰	۳	۱/۲	۰	۱۱/۴	۱/۴	۷٫۲۵
GI	۰	۱/۲	۰	۱/۲	۰	۲	۳/۲	۱/۲	۱۷/۴	۵/۴	۱۰٫۵

شکل نهایی گراف:

عددهایی که کنار یال‌ها نوشته شده است میزان betweenness centrality برای آن یال است.



برای مشخص کردن community های گراف، حداکثر تعداد ۱۰ برای betweenness centrality یال‌ها در نظر می‌گیریم و یال‌هایی که میزان betweenness آن از عدد ۱۰ بزرگ‌تر باشد را حذف می‌کنیم. در نتیجه یال‌های AB و IG و DG و AD و AE حذف می‌شوند و مولفه‌های گراف حاصل در واقع community های گراف را نشان می‌دهند.



2 community دارد.

سوال دوم

الف)

الگوریتم PageRank بر اساس این ایده است که صفحاتی که دارای لینک‌های ورودی بیشتری از صفحات با کیفیت بالا هستند، خود نیز صفحات با کیفیت بالایی هستند. پیاده سازی الگوریتم pagerank با استفاده از random walk میباشد، به این صورت که یک گره فرضی را در نظر می‌گیرد که به‌صورت تصادفی از طریق لینک‌ها به یکی از گره‌های خروجی حرکت می‌کند و یا به صورت تصادفی ب یکی از گره‌های گراف. احتمال اینکه این کاربر به صفحه خاصی برسد، رتبه آن صفحه را تعیین می‌کند.

برای پیاده سازی این الگوریتم ابتدا لیست مجاورت گراف را ایجاد می‌کنیم، در این لیست کلید ها گره های گراف و value ها گره هایی هستند که یالی به آنها از آن گره وجود دارد.

در مرحله بعد، باید یک تابع برای محاسبه pagerank هر گره ایجاد شده است (در ابتدا همه ی صفحات دارای ارزش برابر n/1 هستند و رتبه یک صفحه برابر است با مجموع رتبه‌های صفحاتی که به آن لینک داده‌اند، تقسیم بر تعداد لینک‌های ورودی به آن صفحه.)

```python
def pagerank(adjacency_list, damping_factor=0.85, max_iterations=100, tolerance=1e-6):
    # Initialize variables
    num_nodes = len(adjacency_list)

    initial_pr = 1 / num_nodes
    page_rank = {node: initial_pr for node in adjacency_list}

    # Main iteration loop
    for _ in range(max_iterations):
        new_page_rank = {}
        for node in adjacency_list:
            new_rank = (1 - damping_factor) / num_nodes
            new_rank += damping_factor * sum(
                page_rank[neighbor] / len(adjacency_list[neighbor])
                for neighbor in adjacency_list if node in adjacency_list[neighbor]
            )
            new_page_rank[node] = new_rank

        # Check for convergence
        if all(abs(new_page_rank[node] - page_rank[node]) < tolerance for node in adjacency_list):
            break

        page_rank = new_page_rank

    return page_rank
```

برای گزارش همسایگان مشابه با استفاده از الگوریتم PageRank در گراف جهت دار، باید همسایه های ورودی و خروجی را در نظر بگیریم. زیرا با در نظر گرفتن همسایه های ورودی و خروجی، می توانیم گره هایی را شناسایی کنیم که هم به خوبی به هم متصل هستند و هم الگوی مشابهی از اتصالات دارند.

همسایه های ورودی: با در نظر گرفتن همسایه های ورودی، می توان گره هایی را که معتبر یا تأثیرگذار در نظر گرفته می شوند، شناسایی کرد.

همسایه های خروجی: با در نظر گرفتن همسایه های خروجی، می توانید گره هایی را که مرتبط یا مرتبط با گره فعلی در نظر گرفته می شوند، شناسایی کرد.

الگوریتم PageRank، همسایه های ورودی مهم هستند زیرا می توانند به تعیین اعتبار یک گره کمک کنند. اگر یک گره همسایه های ورودی زیادی داشته باشد که مقادیر PageRank بالایی دارند، احتمالاً گره نیز معتبر در نظر گرفته می شود. همسایه های خروجی نیز مهم هستند زیرا می توانند به تعیین ارتباط موضعی یک گره کمک کنند. اگر یک گره همسایه های خروجی زیادی داشته باشد که در مورد یک موضوع هستند، آنگاه احتمالاً گره مربوط به آن موضوع نیز در نظر گرفته می شود.

در این قسمت لیست همسایگان (گره‌های ورودی و خروجی)گره‌های ورودی محاسبه می شود.

```
neighbor = {}
for s in [688,387,277,876,999,1777,6319]:
    neighbor[s] = graph[s]
    for node in graph:
        if s in graph[node]:
            neighbor[s].append(node)
neighbor
```

در نهایت بین هر گره ورودی، فاصله آن تا همسایگانش محاسبه میشود و ۱۰ نزدیک‌ترین همسایه در خروجی چاپ می‌شوند.

```
688: [226, 229, 854, 1131, 5008, 1531, 1914, 1535, 621, 6097]
387: [1471, 181, 1939, 3667, 3710, 3709, 3713, 3845, 2748, 3711]
277: [224, 854, 1606, 350, 346, 2198, 1744, 2675, 270, 1882]
876: [854, 304, 569, 877, 1015, 885, 2545, 878, 3845, 883]
999: [385, 391, 917, 3785, 2209, 3246, 206, 17, 3806, 2303]
1777: [3220, 2066, 2209, 5008, 3035, 269, 2545, 1251, 2923, 3017]
6319: [7089, 5366, 5424, 6033, 6510, 5312, 7381, 7102, 1549, 1266]
```

در روشی دیگر برای به دست آوردن شباهت دو گره، میانگین pagerank گره‌های ورودی و میانگین pagerank گره‌های خروجی و pagerank همان گره محاسبه شده است.

```
688: [226, 854, 229, 1131, 1531, 5299, 1535, 4865, 1530, 94]
387: [181, 1939, 1471, 3710, 3711, 1156, 3709, 3713, 2459, 3712]
277: [224, 1606, 350, 346, 2675, 1319, 2678, 1124, 2680, 854]
876: [304, 854, 569, 4072, 885, 878, 877, 3195, 3568, 882]
999: [917, 3246, 206, 1689, 17, 3806, 2303, 5079, 4298, 391]
1777: [1251, 2066, 3220, 2248, 509, 719, 269, 2923, 4718, 4367]
6319: [6510, 5424, 4950, 5366, 7102, 7375, 8281, 1159, 4730, 6106]
```

ب)

الگوریتم pagerank می تواند به خوبی همبستگی های کمی بین جفت یا زیر مجموعه گره ها را به تصویر بکشد، به خصوص در گراف‌هایی که در آنها تراکم بسیار کم است ولی می توان از این الگوریتم برای خوشه‌بندی گراف‌ها استفاده کنیم که با شناسایی گره هایی با مقادیر PageRank مشابه و گروه بندی آنها با یکدیگر انجام میشود.

در زیر گام‌های استفاده از pagerank برای خوشه‌بندی گراف ها آمده است:

1. محاسبه pagerank هر گره: الگوریتم PageRank یک مقدار عددی به هر گره اختصاص می دهد که نشان دهنده اهمیت یا مرکزیت آن در شبکه است. مقادیر بالاتر Page Rank مربوط به گره هایی است که در گراف مرکزی و تاثیرگذارتر هستند.
2. شناسایی خوشه ها: می توان با جستجوی گره هایی با مقادیر PageRank مشابه، دسته ها را شناسایی کنیم. گره هایی با مقادیر PageRank مشابه بخشی از یک جامعه یا گروه در گراف هستند.
3. گره ها را در خوشه ها گروه بندی کنید: بر اساس خوشه های شناسایی شده، می توانید گره ها را به گروه ها یا جوامع مختلف اختصاص دهید. گره هایی با مقادیر PageRank مشابه احتمالاً به همان خوشه تعلق دارند.
4. اعتبار سنجی خوشه ها: برای اینکه متوجه بشویم خوشه‌بندی انجام شده نتیجه مطلوبی دارد و یا خیر! این کار را با تجزیه و تحلیل ویژگی های گره ها در هر خوشه و بررسی سازگاری با ساختار مورد انتظار گراف انجام می‌شد.

سوال سوم

الف)

برای پیاده سازی این الگوریتم ابتدا لیست مجاورت گراف را ایجاد می‌کنیم، در این لیست کلید ها گره های گراف و value ها گره هایی هستند که یالی به آنها از آن گره وجود دارد. سپس با استفاده از تابع kosaraju_scc که برای محاسبه component connected strongly در گراف جهت‌دار میباشد. و قطر روی بزرگترین scc گراف محاسبه میشود.

قطر گراف برابر با ۲۲ میباشد و با توجه به اینکه در سوال ذکر شده که مقدار k برابر با جز صحیح تقسیم قطر گراف بر ۲۲ میباشد، حاصل k برابر با ۵ میباشد.

شکل زیر پیاده‌سازی رابطه R داده شده در صورت سوال که برای محاسبه شباهت دو گره می باشد را نشان می‌دهد.

در دیکشنری I برای هر گره، همسایه‌های ورودی آن ذخیره شده است.

```
R={}

for a in nodes:
    for b in nodes:
      if tuple({a, b}) not in R:
        if a!= b:
             R[tuple({a, b})]=0
        else:
            R[tuple({a, b})]=1
```

```
C=0.5
for k in range(1,K+1):
    print(k)
    new_R={}
    for a in nodes:
        for b in nodes:
            len_Ia = len(I[a])
            len_Ib = len(I[b])
            s = 0
            if len_Ia ==0 or len_Ib==0:
                new_R[tuple({a, b})]=0
            else :
               for i in I[a]:
                  for j in I[b]:
                      s+= R[tuple({i,j})]

               new_R[tuple({a, b})] = (C/(len_Ia*len_Ib))*s
    R = new_R
```

بعد از محاسبه شباهت گره‌ها با یکدیگر حاصل آن در فایل similarity.pkl ذخیره شده است.

در تابع زیر برای گره‌های ورودی، همسایگان ورودی و خروجی آنها در لیست همسایگانشان ذخیره میشود.

```
neighbor = {}
for s in [688,387,277,876,999,1777,6319]:
    neighbor[s] = graph[s]
    for node in graph:
        if s in graph[node]:
            neighbor[s].append(node)
```

در نهایت میان هر گره، شباهت آن با لیست همسایگانش در دیکشنری Rank ذخیره می‌شد.

```
Rank ={}
nodes = graph.keys()
for a in [688,387,277,876,999,1777,6319]:
  r={}
  for node in nodes:
    if node in neighbor[a]:
      r[node] = R[tuple({node, a})]
  Rank[a] =r
```

۱۰ شبیه ترین همسایه در شکل زیر برای هر گره نشان داده شده است

```
688: [8555, 7818, 1362, 6569, 1531, 5824, 1386, 5610, 2879, 94]
387: [8446, 1362, 8844, 4004, 2749, 1471, 6514, 6283, 384, 8286]
277: [7553, 1215, 5406, 4695, 5409, 1571, 6842, 390, 614, 1926]
876: [8642, 7962, 8542, 7279, 2723, 2325, 6171, 304, 6365, 883]
999: [6256, 3115, 191, 1378, 5677, 6514, 1979, 8465, 5630, 385]
1777: [191, 6651, 3017, 2209, 2808, 381, 719, 2066, 1958, 1587]
6319: [4163, 6551, 7089, 960, 6271, 737, 810, 3836, 4950, 30]
```

ب)

برای نشان دادن اینکه این الگوریتم خاصیت یکنوایی دارد، باید نشان دهیم که برای هر دو گره a و b، اگر $R_k(a,b)=1$ برای یک تکرار k خاص، پس $R_k(b,a)=1$ نیز برقرار است.

فرض کنید $R_k(a,b)=1$. این بدان معناست که برای هر دو گره i و j در $I(a)$ و $I(b)$، داریم:

$$R_{k-1}(I_i(a),I_j(b))=1$$

از آنجایی که $(R_{k-1}(I_i(a),I_j(b))$ اعداد غیر منفی هستند، هر دو باید برابر با 1 باشند. به عبارت دیگر، برای هر دو گره i و j در $I(a)$ و $I(b)$، داریم:

$$R_{k-1}(I_i(a),I_j(b))=1$$

بنابراین، داریم:

$$1 = 1 \sum_{j=1}^{|I(a)|} \sum_{i=1}^{||I(b)|} \frac{C}{|I(b)||I(a)|} = R_{k-1}(I_i(b), I_j(a)) \sum_{j=1}^{|I(a)|} \sum_{i=1}^{||I(b)|} \frac{C}{|I(b)||I(a)|} = R_k(b,a)$$

نتیجه می گیریم که برای هر دو گره a و b، اگر $R_k(a,b)=1$ برای یک تکرار k خاص، پس $R_k(b,a)=1$ نیز برقرار است. بنابراین، این الگوریتم خاصیت یکنوایی دارد.

یک راه دیگر برای نشان دادن این خاصیت این است که از تعریف شباهت استفاده کنیم. طبق تعریف، شباهت بین دو گره برابر است با احتمال اینکه دو گره از طریق یک مسیر مشترک به هم متصل شوند. از آنجایی که $R_k(a,b)=1$، بنابراین احتمال اینکه دو گره a و b از طریق یک مسیر مشترک به هم متصل شوند برابر با 1 است. این بدان معناست که احتمال اینکه دو گره b و a از طریق یک مسیر مشترک به هم متصل شوند نیز برابر با 1 است. بنابراین، داریم:

$$1 = 1 \sum_{j=1}^{|I(a)|} \sum_{i=1}^{||I(b)|} \frac{C}{|I(b)||I(a)|} = R_{k-1}(I_i(b), I_j(a)) \sum_{j=1}^{|I(a)|} \sum_{i=1}^{||I(b)|} \frac{C}{|I(b)||I(a)|} = R_k(b,a)$$

نتیجه می گیریم که برای هر دو گره a و b، اگر $R_k(a,b)=1$ برای یک تکرار k خاص، پس $R_k(b,a)=1$ نیز برقرار است. بنابراین، این الگوریتم خاصیت یکنوایی دارد.

سوال چهارم

الف)

تابع modularity در تقسیم بندی گراف به صورت زیر تعریف می‌شود:

$$Q = \sum_{s \in S} [(\#\ edges\ within\ group\ s) - (expected\ \#\ edges\ within\ group\ s))]$$

$$Q(G,S) = \frac{1}{2m} \sum_{s \in S} [\sum_{i,j \in s} A_{ij} - \sum_{i,j \in s} \frac{k_i k_j}{2m}] = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} (A_{ij} - \frac{k_i k_j}{2m})$$

رابطه بالا نشان می‌دهد که community ها به اندازه کافی خوب هستند و میتوان از رابطه بالا استفاده کرد و جوامعی را پیدا کرد که رابطه بالا را به حداکثر میرساند، پس داریم:

$$max_S\, Q(G,S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} (A_{ij} - \frac{k_i k_j}{2m})$$

اگر بردار s به صورتی باشد که ۱ به ازای اعضای در خوشه و به ازای بقیه عضوها -۱ تعریف شده باشد، میتوان رابطه بالا را به صورت زیر نوشت:

$$Q(G,S) = \frac{1}{2m} \sum_{i \in N} \sum_{j \in N} (A_{ij} - \frac{k_i k_j}{2m})(\frac{s_i s_j + 1}{2}) = \frac{1}{4m} \sum_{i \in N} \sum_{j \in N} (A_{ij} - \frac{k_i k_j}{2m}) s_i s_j$$

ب)

G : گراف ورودی که عمل تشخیص اجتماع بر روی آن انجام می‌شود.

m: تعداد کل یال‌ها درون گراف

A : ماتریس مجاورت گراف را نشان میدهد که درایه‌های آن برابر صفر و یا یک است

K_i : درجه گره i را در گراف نشان می‌دهد که برابر با جمع سطر متناظر با آن در ماتریس مجاورت است.

S : بردار عضویت در community را نشان می‌دهد، برای مثال برای تقسیم شبکه به دو گروه، اگر راس i متعلق به گروه 1 است، si = 1 و اگر به گروه 2 تعلق دارد si = -1.

ج)

شبکه حاوی n راس است. برای تقسیم خاصی از شبکه به دو گروه، اگر راس i متعلق به گروه 1 است، si = 1 و اگر به گروه 2 تعلق دارد si = -1. و تعداد یال های بین رئوس i و j را Aij در نظر بگیرید، که معمولا 0 یا 1 باشد، تعداد یال های مورد انتظار بین رئوس i و j اگر یال ها به طور تصادفی قرار گیرند kikj/2m است، جایی که ki و

kj درجاتی از رئوس و m = ½Σi ki تعداد کل یال های شبکه است. با مشاهده اینکه مقدار ½ (1 sisj + 1) است اگر i و j در یک گروه باشند و 0 در غیر این صورت، می توانیم مدولار بودن را به صورت زیر بیان کنیم:

$$Q = \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1) = \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j,$$

مقدار B به صورت زیر تعریف می‌شود:

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m},$$

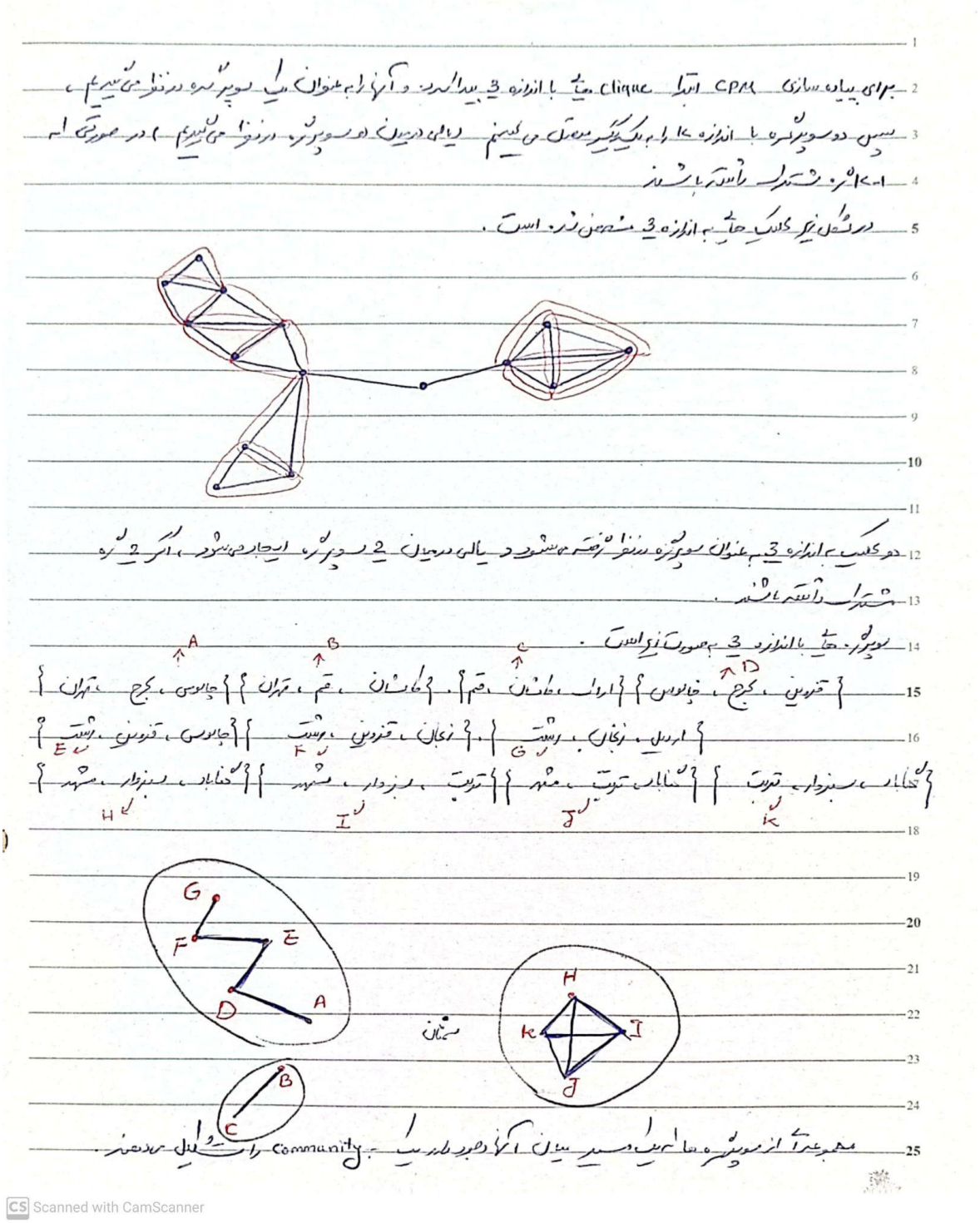
بنابراین داریم:

$$Q(G,S) = \frac{1}{2m} \sum_{i \in N} \sum_{j \in N} (A_{ij} - \frac{k_i k_j}{2m})(\frac{s_i s_j + 1}{2}) = \frac{1}{4m} \sum_{i \in N} \sum_{j \in N} (A_{ij} - \frac{k_i k_j}{2m}) s_i s_j$$

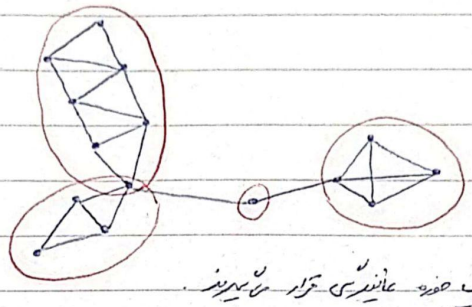
$$= \frac{1}{4m} \sum_{i \in N} \sum_{j \in N} B_{ij} s_i s_j$$

$$= \frac{1}{4m} \sum_{i \in N} s_i \sum_{j \in N} B_{ij} s_j = \frac{1}{4m} s^T B s$$

سوال پنجم



در سوپرشبکه‌های به‌دست آمده از مرحله‌ی قبل بر اساس ارتباط‌ها و میزان آن سوپرشبکه‌ها با یکدیگر ترکیب شده و اجتماع برگشته‌ها تشکیل دادند، یا به‌عبارت دیگر Community‌ها نهایی به‌صورت زیر است:



دقت Community‌های به‌دست‌آمده در این حوزه عالیتری قرار می‌گیرند.

Community 1: { سمنان }

Community 2: { ایران، قم، تهران، کاشان }

Community 3: { تربت حیدریه، کابل، مزار، سبزوار }

Community 4: { اردبیل، زنجان، قزوین، رشت، چالوس، کرج، تهران }

سوال ششم

الف)

مراحل اصلی روش spectral clustering به صورت زیر می‌باشد:

۱- در ابتدا برای گراف ماتریس مجاورت و سپس ماتریس (تنها عناصر قطر اصلی غیر صفر هستند و درجه‌ی هر گره را مشخص میکند)درجه ساخته میشود.

۲- ایجاد ماتریس لاپلاسین برای گراف (ماتریس مربعی که از تفاضل دو ماتریس درجه و ماتریس همسایگی ساخته می شود.)

۳- محاسبه مقدارویژه و بردار ویژه ماتریس لاپلاسین (مقادیر ویژه نشان دهنده اهمیت نسبی هر بعد از فضای Eigen هستند. بردار ویژه نیز نشان دهنده جهت هر بعد از فضای Eigen است).

۴- سورت کردن مقادیر ویژه ماتریس لاپلاسین و استخراج بردار ویژه و مقدار ویژه متناظر با دومین مقدار ویژه سورت شده (بردار ویژه دوم نمایانگر کاهش بعد یافته هر گره ماتریس می‌باشد).

۵- استفاده از یک الگوریتم خوشه بندی سنتی برای خوشه بندی بردار ویژه استخراج شده است.

ب)

مراحل خوشه بندی به این صورت است:

ابتدا ماتریس لاپلاسین را برای گراف مربوطه حساب میکنیم. برای محاسبه ماتریس لاپلاسین از رابطه زیراستفاده می شود:

$$L = D-I$$

که در آن A ماتریس مجاورت گراف و D یک ماتریس قطری است که هر عنصر روی قطر درجه نود متناظر با آن را نشان می دهد.برای مثال]D[3,3 درجه نود 3 را نشان می دهد.

ماتریس لاپلاسین استخراج شده به این شکل است:

```
array([[ 6., -1., -1., ...,  0.,  0.,  0.],
       [-1.,  4.,  0., ...,  0.,  0.,  0.],
       [-1.,  0.,  4., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  3.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  7.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  5.]])
```

بعد از محاسبه ماتریس لاپلاسین، مقادیر و بردارهای ویژه را برای آن محاسبه کرده و مرتب می‌کنیم. دومین کوچکترین بردار ویژه را استخراج کرده و از آن به عنوان بردار ویژگی برای خوشه بندی استفاده میکنیم. به این صورت که هر عنصر بردار lambda2 (بردار ویژه ی متناظر با دومین کوچکترین مقدار ویژه)متناظر با یکی از نودهای گراف است .

```
### obtain the eigenvalues and eigenvectors of the laplacian
eigenvalues, eigenvectors = np.linalg.eig(L)
```

```
index = np.argsort(eigenvalues)
sorted_eigenvalues = eigenvalues[index]
sorted_eigenvectors = eigenvectors[:,index]
```

کد بالا برای محاسبه مقدار ویژه و بردار ویژه ماتریس لاپلاسین و مرتب کردن آن میباشد.

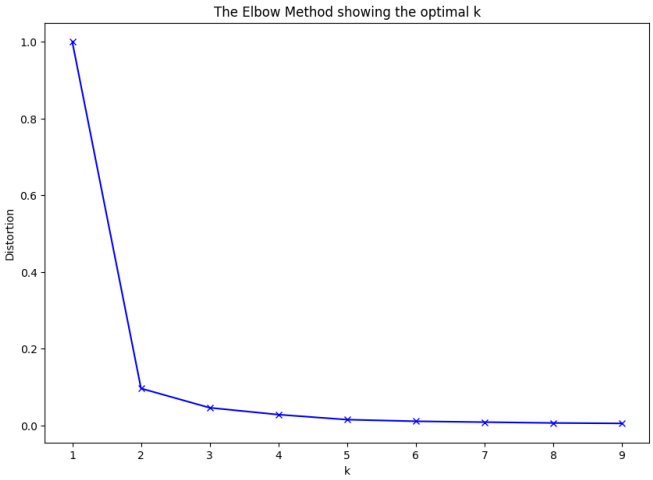
```
l2 = sorted_eigenvalues[1]
x2 = sorted_eigenvectors[:,1]
```

```
x2 = np.squeeze(np.asarray(x2))
x2_real = [np.real(a) for a in x2]
```

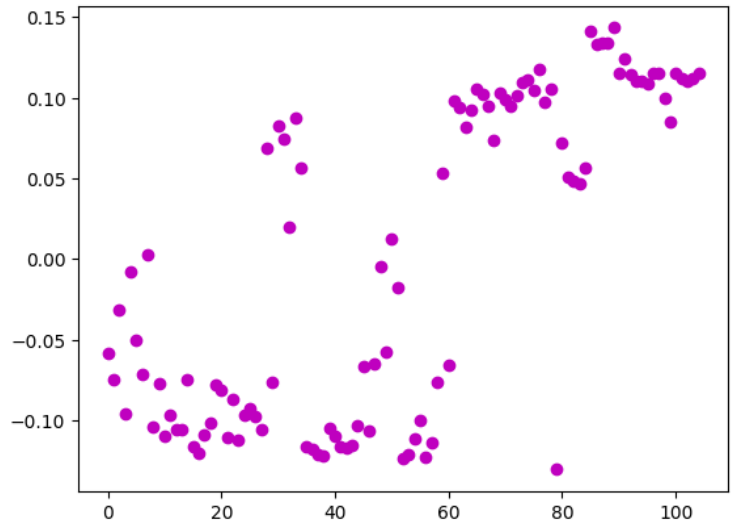
کد بالا برای استخراج بردار ویژه متناظر با دومین کوچکترین مقدار ویژه میباشد و از آن به عنوان کاهش بعد نودهای گراف استفاده میشود.

ج)

برای یافتن تعداد مناسب خوشه هااز روش elbow در الگوریتم k-means استفاده شد. نمودار روش elbow در شکل زیر رسم شده است:



روش دیگر برای اطمینان از تعداد مناسب خوشه ها رسم داده ها در فضای یک بعدی است. شکل زیر نمودار داده‌ها را نشان می‌دهد:



همانطورکه مشاهده می شود تعداد بهینه برای خوشه‌ها ۴ عدد می باشد زیرا تا ۴ خوشه کاهش مقدار distortion خیلی زیاد بوده اما از آن به بعد این کاهش محسوس نیست. بنابراین تعداد ۴ خوشه را برای الگوریتم k-means انتخاب می کنیم.

تابع زیر مقدار modularity را محاسبه میکند، که براساس رابطه سوال قبل میباشد:

```
def modularity(clusters,A,num_edges):
  sum = 0
  for cluster in clusters:
    for i in cluster:
      for j in cluster:
        sum+= (A[i][j] - (D[i][i] * D[j][j])) / (2* num_edges)
  sum/= (2* num_edges)
  return sum
```

تابع زیر مقدار min-cut را محاسبه میکند (برای محاسبه این معیار باید تعداد یال هایی که یک سرشان در یک کلاستر و سر دیگرشان در کلاستر دیگری است را بشماریم.).

```
def min_cut(clusters):
  coms = list(itertools.combinations(clusters, 2))
  sum = 0
  for com in coms:
    for i in com[0]:
      for j in com[1]:
              sum+=A[i][j]

  return sum
```

در تصویر زیر مقدار min-cut و modularity برای k متفاوت محاسبه شده است:

```
num_clusters:1: min_cut:0 modularity:-0.9988662131518167
num_clusters:2: min_cut:20.0 modularity:-0.4989202030018074
num_clusters:3: min_cut:43.0 modularity:-0.41442351695022117
num_clusters:4: min_cut:90.0 modularity:-0.33319964418116976
num_clusters:5: min_cut:138.0 modularity:-0.27731757858093614
num_clusters:6: min_cut:171.0 modularity:-0.22892981833700451
num_clusters:7: min_cut:182.0 modularity:-0.2176613653775933
num_clusters:8: min_cut:234.0 modularity:-0.15666826065271094
num_clusters:9: min_cut:256.0 modularity:-0.14331734205397947
```

با توجه به مقادیر min-cut و modularity در شکل بالا انتخاب ۴ کلاستر، انتخاب مناسبی است. کلاستر های ایجاد شده به صورت زیر می باشد.

```
[ 28  30  31  33  61  62  63  64  65  66  67  68  69  70  71  72  73  74
  75  76  77  78  80  85  86  87  88  89  90  91  92  93  94  95  96  97
  98  99 100 101 102 103 104]
[ 3  8 10 11 12 13 15 16 17 18 21 23 24 25 26 27 35 36 37 38 39 40 41 42
 43 44 46 52 53 54 55 56 57 79]
[ 4  7 32 34 48 50 51 59 81 82 83 84]
[ 0  1  2  5  6  9 14 19 20 22 29 45 47 49 58 60]
```

د)

بله، می توان از روش k-means مستقیماً برای خوشه‌بندی گراف استفاده کرد. اما این کار ممکن است نتایج مطلوبی نداشته باشد. (روش k-means یک روش خوشه بندی است که بر اساس مفهوم فاصله بین داده ها عمل می کند. این روش ابتدا مرکز خوشه هایی را به صورت تصادفی انتخاب می کند و سپس داده ها را به نزدیکترین مرکز خوشه اختصاص می دهد. سپس، مراکز خوشه ها به گونه ای اصلاح می شوند که داده ها به طور متوسط به مرکز خوشه خود نزدیکتر باشند. این فرآیند تا زمانی که مراکز خوشه ها دیگر تغییر نکنند، تکرار می شود.)

می توان از همان ابتدا روش k-means را بر روی گراف اجرا کرد و با استفاده از آن گراف را بخش بندی کرد. اما ماتریس Laplacian مزایای زیر را دارد:

- توانایی شناسایی خوشه های غیر محدب: روش k-means فقط می تواند خوشه های محدب را شناسایی کند. اما ماتریس Laplacian می تواند خوشه های غیر محدب را نیز شناسایی کند.
- استفاده از ساختار گراف: روش k-means از ساختار گراف استفاده نمی کند. اما ماتریس Laplacian از ساختار گراف استفاده می کند. این امر باعث می شود که روش spectral clustering نسبت به روش k-means برای کاربردهای مبتنی بر گراف مناسب تر باشد. زیرا در روش spectral clustering ابتدا ماتریس لاپلاسین را برای گراف محاسبه می کند و سپس بردار ویژه های ماتریس لاپلاسین را استخراج می کند. این بردار ویژه ها نشان دهنده موقعیت داده ها در فضای Eigen هستند. سپس، روش k-means بر روی فضای Eigen اجرا می شود تا داده ها به خوشه ها اختصاص داده شوند.
- مقاومت در برابر نویز: روش k-means نسبت به نویز و موارد غیر معمول حساس است. اما ماتریس Laplacian نسبت به نویز و موارد غیر معمول مقاوم تر است.

روش k-means نمی تواند خوشه غیر محدب را شناسایی کند و آن را به یکی از دو خوشه محدب دیگر اختصاص می دهد. اما ماتریس Laplacian می تواند خوشه غیر محدب را شناسایی کند و آن را به یک خوشه جداگانه تبدیل کن و اگر گراف شامل دو خوشه باشد که با یک لبه باریک از یکدیگر جدا شده اند. روش

k-means ممکن است این دو خوشه را به عنوان یک خوشه واحد شناسایی کند. اما ماتریس Laplacian می تواند این دو خوشه را به عنوان دو خوشه جداگانه شناسایی کند.

سوال هفتم

الف)

روش‌های AGM,CPM همپوشانی جوامع را در نظر می‌گیرند ولی روش‌های Grivan-Network, Fast Modularity, Spectral Clustering همپوشانی جوامع را در نظر نمی‌گیرند.

روش Grivan-Network: در این روش برای تک‌تک یالها مقدار betweenness centrality را محاسبه میکنیم و سپس یال با بالاترین مقدار را حذف میکنیم و این فرآیند را ادامه میدهیم تا به cluster مطلوب برسیم، در این روش از قدرت یال‌های ضعیف برای تشخیص خوشه‌ها استفاده می‌کند و با حذف آنها در واقع جواب را به صورت غیر هم پوشان در نظر میگیرد (در این روش فرض بر این است که یالهایی با betweenness centrality بزرگتر، دو خوشه را به یکدیگر متصل کرده اند و با حذف آن خوشه‌ها از یکدیگر جدا شده و در نتیجه خوشه‌ها به صورت یر هم‌پوشان در نظر گرفته‌شده‌اند).

روش Modularity : در این روش برای ماتریس sj به دست آمده، را بطه زیر در نظر گرفته می‌شود:

- $$s_j = \begin{cases} +1 & if\ x_{n,j} \geq 0 \quad (\text{j−th coordinate of } x_n \geq 0) \\ -1 & if\ x_{n,j} < 0 \quad (\text{j−th coordinate of } x_n < 0) \end{cases}$$

با توجه به این رابطه یک گره نمیتواند برای دو خوشه باشه، پس این روش نیز همپوشانی را در نظر نگرفته است.

روش Spectral Clustering : در این روش بعد از به دست‌آوردن ماتریس لاپلاسین و حاسبه‌ی مقادیر ویژه و بردار ویژه برای آن، بردار ویژه مناظر با مقدار ویژه دوم در نظر گرفته می‌شود و از این روش برای کاهش بعد گره‌ها درون گراف استفاده می‌شود (بردار ویژه روم) و سپس یک روش خوشه بندی روی دیتای کاهش بعد یافته اعمال می‌شود. در این روش با توجه به اینکه داده کاهش بعد یافته خوشه بندی می‌شود و حتی در راهکار میتوان ۰ را آستانه در نظر گرفت و براساس آن گراف را در هر مرحله خوشه بندی کرد، در این روش همپوشانی در نظر گرفته نمی‌شود.

روش CPM: در این روش که همپوشانی خوشه‌ها در نظر گرفته می‌شود، ابتدا بزرگترین k clique ها درون گراف به عنوان ابر گره در نظر گرفته میشوند و سپس این کلیک ها با یکدیگر ترکیب میشوند در صورتی که k-1 گره مشترک داشته باشند. با توجه به اینکه در این روش در کلیک ها امکان اشتراک گره وجود دارد همپوشانی در نظر گرفته می‌شود.

روش AGM: این روش که برای خوشه بندی گراف های دنیای واقعی کاربرد دارد، فرض بر این است که در ناحیه اشتراک کلاسترها اجتماع و چگالی تعداد یالها بیشتر است.

ب)

در روش AGM فرض بر این است که در محل همپوشانی جوامع مختلف تراکم یال ها بیشتر از تراکم در هر یک از جوامع است. در این روش پارامترای ({V,C,M,{pc) وجود دارد (V تعداد گره‌ها، C تعداد جوامع، M تابع عضویت است که مشخص میکند هر گره به گره به کدام کلاستر تعلق دارد و pc احتمال یال بین گره‌ها را برای هر جامعه مشخص میکند) در این روش گرافی تولید می‌شود که با افزایش کلاسترها چگالی آن قسمت از گراف بیشتر شود و با این روش خوشه ها را پیدا میکند.

$$p(u,v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c)$$

این روش که فرض بر این است که گراف‌های دنیای واقعی براساس این مدل تولید شده‌اند و سعی میشود با این فرض پارامترای این الگوریتم را از گراف دنیای واقعی یاد بگیرد.

$$\arg\max_B P(G \mid B) = \prod_{(i,j) \in E} P(i,j) \prod_{(i,j) \notin E} (1 - P(i,j))$$