دانشگاه صنعتی امیرکبیر دانشکده مهندسی کامپیوتر

مهندسی کامپیوتر گرایش نرم افزار

گزارش تمرین اول

شبکه پیچیده

توسط:

روح الله احمدی

نیمسال اول سال تحصیلی ۱۴۰۱

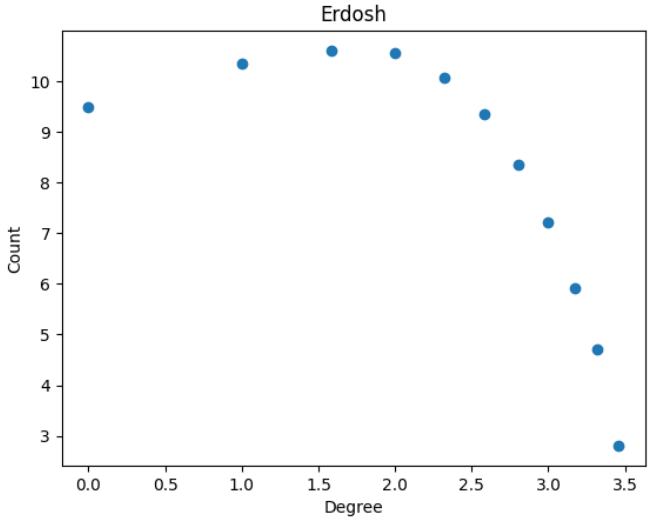
سوال اول :

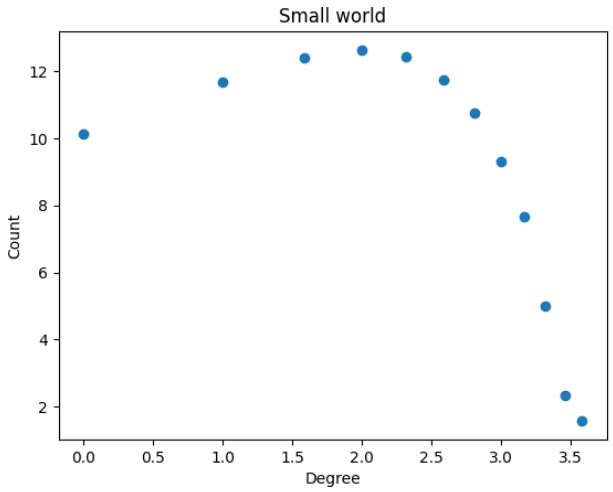
الف) برای ایجاد گراف erdosh-reiny ابتدا به تعداد یالها که در صورت تمرین مشخص شده بود ایجاد کردیم. سپس با احتمال ۰.۷ یالها را ذخیره میکنیم. برای حذف یالهای تکراری از دیتااستراکچر set استفاده میکنیم. برای گراف ایجاد شده یک بار لیست نودها و یک بار درجه آن را ذخیره میکنیم.

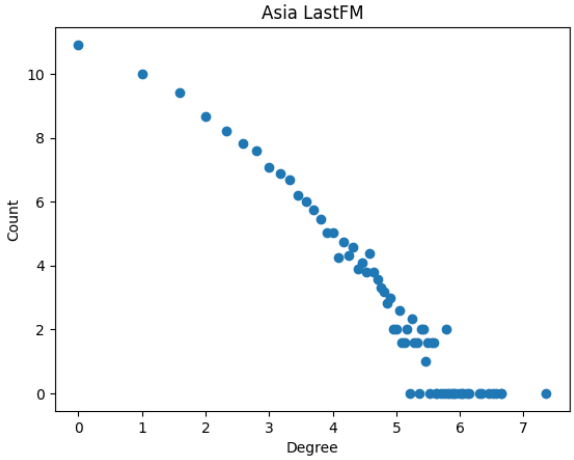
ب) برای ایجاد گراف small world ابتدا یک گراف ring می سازیم. هر نود را به چند نود بعدی متصل کرده و در از راه حل rewire با احتمال ۰.۷ یالها را حذف کردیم. درنهایت گراف small world بوجود می آید. برای این گراف نیز یک لیست نودها و یک دیکشنری از همسایه ها ایجاد میکنیم.

ج) دیتاست LastFM را دانلود کرده و دو دیتا استراکچر برای لیست نودها و همسایگی نودها که یک دیکشنری با کلید نام نود و مقدار همسایه ها ایجاد کردیم. برای جلوگیری از تکرار نیز از دیتااستراکچر set استفاده نمودیم.

د) گرافهای مربوط به سه گراف erdosh و small world و lastfm را ایجاد می کنیم .

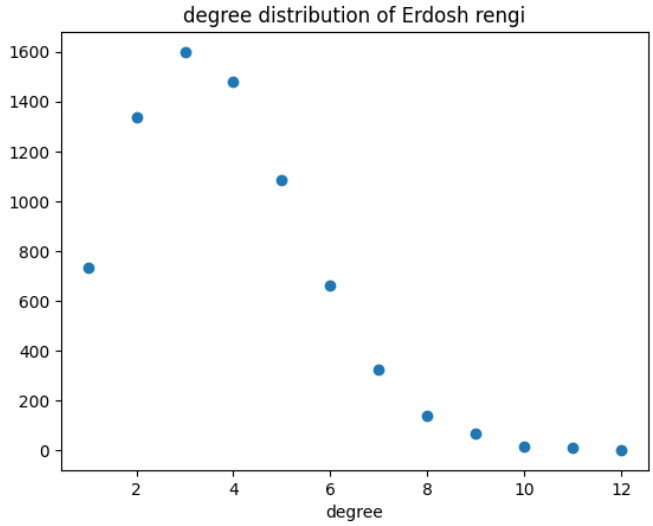


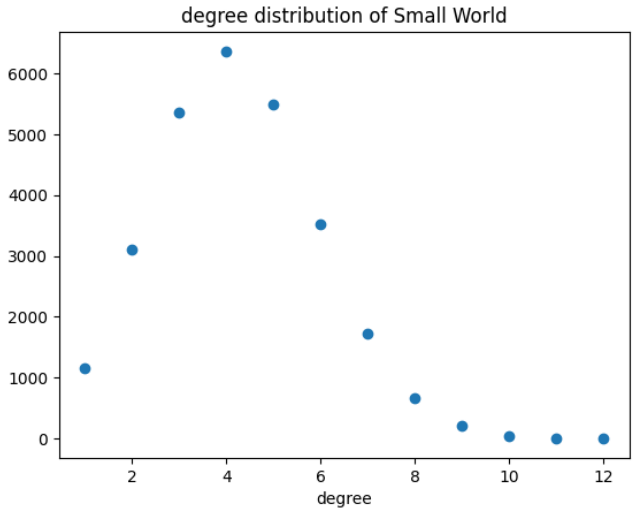


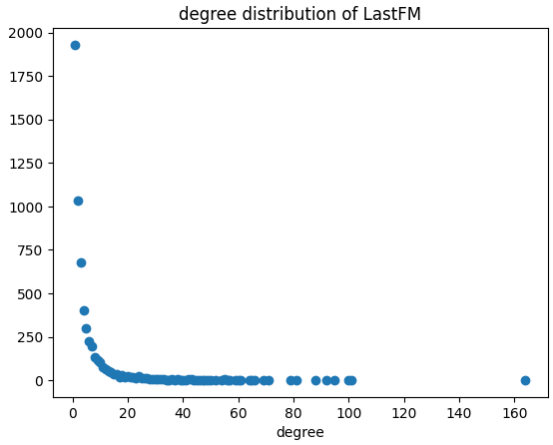


گراف lastFM که یک گراف دنیای واقعی است دارای یک دم کلفت است که به این معنی است که نودهایی با درجه بالا تعدادشان زیاد است. بنابراین این گراف یک گراف پیچیده واقعی است. این ویژگی ها در گرافهای ساختگی صادق نیست.

نمودار توزیع درجه نیز به شکل زیر می باشد:







همانطور که می بینید گرافهای ساختگی دارای توزیع درجه به شکل زنگوله هستند در حالی که در گراف دنیای واقعی این گونه نیست.

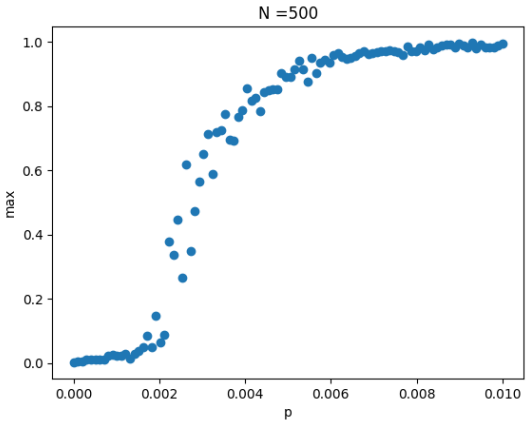
ه) کلاستر کوافیشنت در سه گراف بالا به مقدار زیر است:

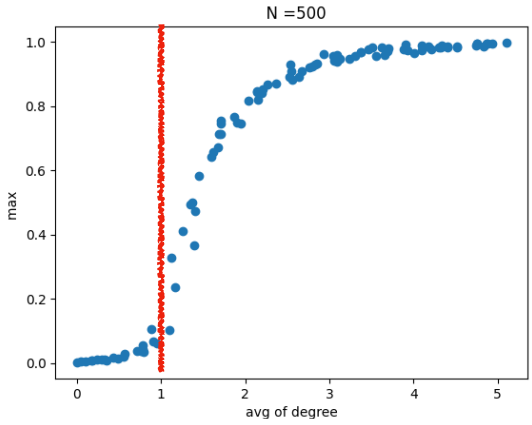
```
Cluster coeffient erdos 0.0015840390102007326
Cluster coeffient small world 0.21602919357356354
Cluster coeffient asia 0.219418424327086
```

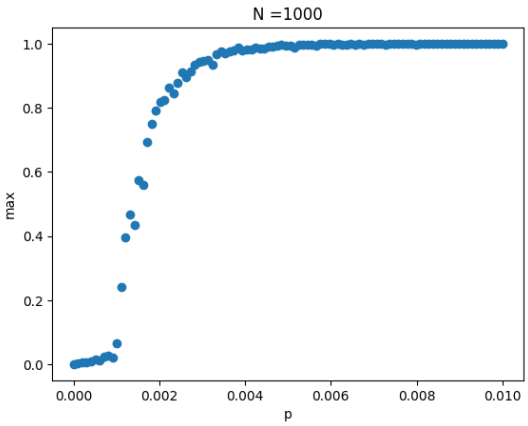
گراف اردوش رنینگ پایینترین مقدار در کلاستر کوافیشنت را دارد. پس از آن گراف small world است و در نهایت گراف دنیای واقعی. البته قابل یادآوری است که گراف small world وابسته به مقدار احتمالاتی p است و با کم و زیاد کردن این مقدار ممکن است جایگاه گراف دنیای واقعی و small world جابجا شود ولی در نهایت عدد کلاستر کوافیشنت برای این دو نمودار نزدیک هم می باشد.

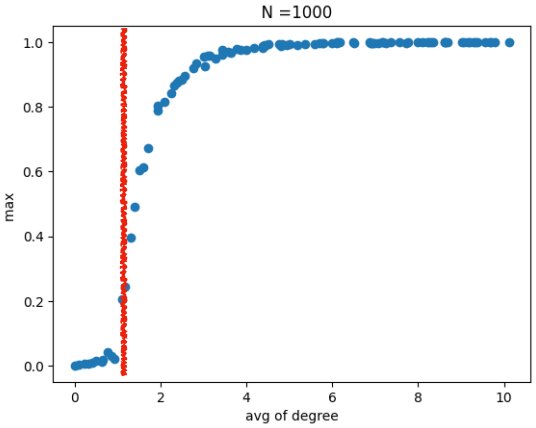
سوال دوم)

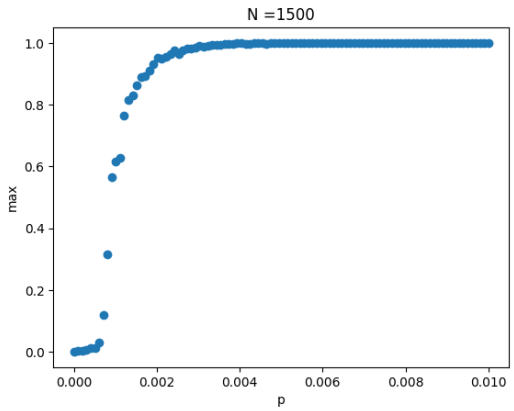
برای نمایش بزرگترین کامپوننت ابتدا یک iteration روی گراف انجام میدهیم و بزرگترین کامپوننتی که در این iteration دیده ایم را نگه داری میکنیم. نمودار میانگین درجه و اندازه بزرگترین کامپوننت را رسم میکنیم. خط قرمز نشان دهنده یک جاینت کامپوننت است.

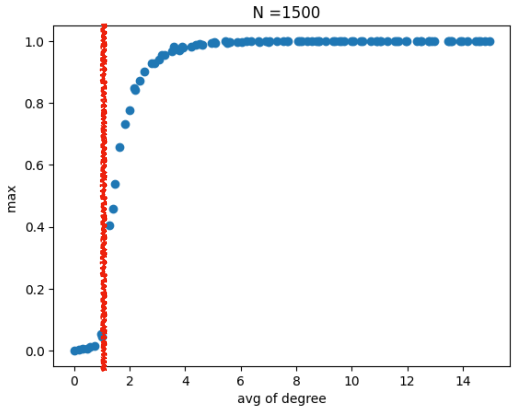












در صورتی که p که میدهیم کمتر از ۱/n باشد میانگین درجات کمتر از یک می باشد. یعنی نود ایزوله در گراف داریم. تقریبا یالی تشکیل نشده. وقتی احتمال بزرگتر از logn بر n شود مولفه همبند در گراف نداریم و همه گراف متصل است.

سوال سوم)

الف) در ساختار دو بخشی N۱*N۲

ب) لینک هایی که نیمتوانند با حالت دو بخشی نبودن این گراف اتفاق بیوفتند همان یال داخلی هر بخش است که برابر است با

ج) چگالی با فرض تعریف سوال عبارت زیر خواهد بود

$$\frac{N_2 * N_1}{\binom{N_1 + N_2}{2}} = \frac{N_2 * N_1}{(N_2 + N_1) * (N_2 + N_1) * 0.5} = \frac{2 * N_2 * N_1}{{N_2}^2 + 2 * N_2 * N_1 + {N_1}^2} = \frac{2 * N_2 * N_1}{{N_2}^2} = \frac{2 * N_1}{N_2}$$

د) مطابق میانگین درجه که برابر است با تعداد یالها به تعداد گره ها

برای بخش ۱ :

$$\frac{N_2 * N_1}{N_1}$$

برای بخش ۲ داریم :

$$\frac{N_2 * N_1}{N_2}$$

درنتیجه

$$\frac{k_1}{k_2} = \frac{N_2}{N_1}$$

سوال چهارم) در این سوال ابتدا الگوریتم Gready پیاده سازی شده که در آن بیشترین تاثیرگذاری برای همه نودها محاسبه شده و پس نود با بیشترین تاثیر گذاری به عنوان عضو اول مجموعه S در نظر گرفته می شود. دوباره این محاسبه بیشترین تاثیر گذاری برای همه نودها حساب می شود و نود با بیشترین تاثیر به عنوان عضو دوم مجموعه S انتخاب می شود. این پروسه تا پایان ۱۰ نود درخواستی انجام می پذیرد.

در الگوریتم Lazy hill climing ابتدا یک محاسبه برای همه نودها شده و بهترین نود انتخاب می شود. سپس نودها با توجه به امتیازشان Sort شده و نود بعدی را بر میداریم . مقدار آن را آپدیت کرده و با نود بعدی مقایسه میکنیم. درصورتی که از مقدار بعدی بیشتر بود به عنوان نود بعدی در مجموعه S قرار میدهیم. درغیر این صورت نود را آپدیت کرده و در جایگاه صحیح اش قرار میدهیم و نود سوم را برمیداریم و این پروسه را تا آخرین نود تکرار خواهیم کرد.

در روش benefit cost روش محاسبه benefit تغییر کرده و از تقسیم out بر روی cost برای امتیاز گذاری استفاده می شود. قابل یادآوری است که با توجه به این که مقدار هزینه در این سوال ۱ در نظر گرفته شده است نتایج سه مدل تقریبا یکی خواهد بود.

تفاوت دو روش greedy و lazy:

الگوریتم greedy ابتدا از یک مجموعه تهی برای S شروع میکند و در تکرار اول نودی را انتخاب میکنیم که بیشترین اندازه out را دارد . سپس در هر تکرار فرض میکنیم که اعضا اول تا ۱-l مجموعه S را پیدا کردیم. برای انتخاب نود l نودی را انتخاب میکنیم که بین همه نودهای اضافه نشده به S بیشترین مارجینال گین را داشته باشد

$$\max_{u} f(S_{i-1} \cup \{u\})$$

در نهایت مقدار F(s) را برای هر u ای که بیشتر شد ان را انتخاب میکنیم وبه S اضافه میکنیم.

شبه کد الگوریتم greedy:

Algorithm:

- Start with $S_0 = \{\}$
- For $i = 1 \dots k$
  - Activate node u that max $f(S_{i-1} \cup \{u\})$
  - Let $S_i = S_{i-1} \cup \{u\}$

الگوریتم celf

بخش اول این الگوریتم از ایده ای برای افزایش سرعت استفاده کرده یعنی هرچقدر iteration جلو می رود marginal gain نودها بیشتر نمی شود بلکه کاهش میابد یا ثابت میماند. از این نتیجه گیری می توانیم به این شکل استفاده کنیم : اگر نودی مثل v داشته باشیم و marginal gain آن را در iteration جدید حساب کنیم و مشاهده کنیم که مقدار آن از marginal gain نودهای دیگر در iteration قبلی بیشتر است در این صورت می توانیم نتیجه بگیریم که marginal gain نودv در iteration جدید نیز مقدارش از بقیه بیشتر است چون هر چه iteration جلو میرود marginal gain نودها کاهش پیدا میکند.

نتایج بدست آمده شامل موارد زیر می باشد:

```
greeedy time  0:00:00.166431

[3, 99974, 38863, 67720, 65252, 33128, 358, 131978, 100471, 885]
```

```
lazy hill climbing time  0:00:00.012762

[3, 99974, 38863, 65252, 67720, 33128, 358, 131978, 100471, 885]
```

```
benefit cost time  0:00:00.029363

[3, 99974, 38863, 65252, 67720, 33128, 358, 131978, 100471, 885]
```

همانطور که مشاهده می شود زمان در روش greedy مقدار ۱۶ میلی ثانیه می باشد که از دو روش دیگر ۰.۰۱۲ میلی ثانیه در روش hill climbing و ۰.۰۲۹ در روش benefit cost بشدت بیشتر است

سوال پنجم)

در این سوال سه مدل Gready و lazy hill climbing و benefit cost را پیاده سازی کردیم. در این سوال n ده در نظر گرفته شده است. برای افزایش سرعت از دیکشنری برای ذخیره همسایه ها و set برای ذخیره یالها در نظر گرفته شده است. این دیتا استراکچر باعث افزایش سرعت می شود.

فرض) ابتدا یک گراف realization می سازیم. سپس ده outbreak را در شبکه پخش میکنیم و از سه مدل یاد شده برای پیدا کردن بهترین نودها برای قراردادن sensor استفاده میکنیم. ابتدا امتیاز گره ها نسبت به out break ها را مشخص کرده و سنسور اول که نودی بهترین امتیاز است و گره outbreak را از گراف حذف میکنیم. مرحله دوم دنبال بهترین نود با بهترین امتیاز برای شناسایی outbreak دوم محاسبه میکنیم تا آخرین outbreak شناخته شود.

نتایج بدست آمده برای سه روش :

```
hill climbing time 0:01:17.636883
13 -20
100450 -19
6947 -19
40000 -19
81761 -19
61429 -403
65540 -20
81925 -20
49159 -20
98315 -20
49163 -20
```

```
{100450: -19, 6947: -19, 40000: -19, 81761: -19, 129998: -19, 65540: -20, 98315: -20, 13: -20, 15: -20, 98325: -20}
lazy hill climbing time :  0:00:00.031425
```

```
{128081: 1803.75, 53213: 1774.6875}
benefit cost time:  0:00:00.004840
```

در نهایت الگوریتم celf که از ماکسیزمم S’ و S’’ بدست می آید مجموعه زیر است :

```
128081
53213
40000
81761
129998
65540
98315
13
15
98325
```

با توجه به زمان ها به راحتی قابل درک است که الگوریتم حریصانه زمان بسیار بیشتری نسبت به الگوریتم celf گرفته است. این عدد برای حریصانه ۱.۲۷ دقیقه ولی برای الگوریتم lazy و benefit کمتر از یک ثانیه می باشد که نشان از سرعت بالای محاسبه در این الگوریتم هاست.

قابل یادآوری است اعداد نشان داده شده در کنار id نود ها بیانگر امتیاز نودها می باشند.