# گزارش پروژه

نیم‌سال دوم ۱۴۰۱-۱۴۰۲

دانشکده علوم ریاضی

اعضای گروه: مهرشاد تازیکی و محمدامین رئیسی

**چکیده**

مساله‌ی Densest-Subgraph، کاربردهای متعددی در زمینه‌های استخراج داده از گراف، مخابرات و شبکه‌های اجتماعی، بیولوژی و علوم مالی دارد. این مساله، به صورت مجرد، به دنبال پیدا کردن یک زیرگراف از یک گراف با بیش‌ترین چگالی ممکن است. برای مفهوم چگالی یک گراف، روابط متعددی وابسته به کاربرد می‌توان ارائه کرد؛ اما در ساده‌ترین حالت، می‌توان چگالی یک زیرمجموعه‌ی $S$ از گراف را برابر $\frac{|E[S]|}{|S|}$ تعریف کرد. این حالت از مساله در سال‌های اخیر مورد توجه بسیاری قرار گرفته است و الگوریتم‌های دقیق و تقریبی متعددی برای آن ارائه شده‌است.
در این گزارش، ابتدا معروف‌ترین الگوریتم‌های این حالت از مساله و هم‌چنین برخی از حالت‌های دیگر مساله را به تفصیل شرح داده‌ایم و به صورت عملی، این الگوریتم‌ها را از نقطه‌نظرهای متفاوتی روی دیتاست جامعی متشکل از 77 تست‌کیس‌های تصادفی و غیر تصادفی (که گراف‌های معروف در دنیای واقعی هستند) مورد مطالعه و تحلیل قرار داده‌ایم. نتایج به دست‌آمده توسط ما، یافته‌های پیشین از منظر تئوری از جمله ضریب تقریب الگوریتم‌های تقریبی داده‌شده و همگرایی آن‌ها را تصدیق می‌کند و هم‌چنین مشاهدات جدیدی را ارائه می‌کنیم که می‌توان در آینده آن‌ها را مورد تحقیق قرار داد.

## ۱ مقدمه

مساله‌ی چگال‌ترین زیرگراف، یکی از مسائل بنیادی با کاربردهای متعدد در دنیای واقعی از جمله مخابرات و استخراج داده است. در این مساله، ورودی یک گراف ساده‌ی $G = (V, E)$ است و هدف، پیدا کردن یک زیرمجموعه $S \subseteq V$ است که $d(S)$ بیشینه باشد. $d(S)$ که چگالی زیرگراف القایی روی رئوس $S$ است، با توجه به ادبیات مد نظر، می‌تواند تعاریف مختلفی داشته باشد. ساده‌ترین و معروف‌ترین، تعریف برای $d(S)$، درجه‌ی متوسط رئوس زیرگراف القایی $S$ یا معادلا $\frac{|E[S]|}{|S|}$ می‌باشد. البته به جز این تعریف استاندارد، تعاریف دیگری مانند چگال‌ترین زیرگراف جهت‌دار [۱۸]، یا چگال‌ترین زیرگراف از نظر تعداد خوشه‌ها [۱۹] وجود دارد.
یک تعمیم کلی از ساده‌ترین حالت، قرار دادن $d(S) = \frac{f(S)}{|S|}$ است که $f(S)$ یک تابع ابرپیمانه‌ای با خواص جزئی‌ای است که این خواص به تفصیل در گزارش اشاره شده‌اند. از مطالب کلاس، می‌دانیم که $E[S]$ یک تابع ابرپیمانه‌ای است و در نتیجه، حالت ساده‌ی ارائه‌شده، حالت خاصی ازین مساله‌ی کلی می‌باشد. نکته‌ی جالبی که در مورد حالت ساده‌ی مساله و هم‌چنین تعمیم آن به ازای توابع ابرپیمانه‌ای وجود دارد، این است که این مسائل در کلاس پیچیدگی $\mathcal{P}$ قرار دارند اما از آن‌جایی که این الگوریتم‌ها روی گراف‌های بزرگ و حجیم استفاده می‌شوند، پیدا کردن الگوریتم‌های تقریبی با زمان اجرای سریع‌تر یکی از کارهایی است که در چند سال اخیر مورد اهمیت ویژه‌ای قرار گرفته است.

### ۱.۱ کارهای پیشین

برای حالت ساده‌ی این مساله، اولین الگوریتم را Picard و Queyranne در سال 1982 بر پایه‌ی محاسبات شار ارائه کردند. [۲۰] پس از 2 سال، Goldberg الگوریتم آن‌ها را از نظر پیچیدگی زمانی بهبود داده است [۱۳] که تا کنون سریع‌ترین الگوریتم دقیقی است که برای این مساله ارائه شده‌است. به مدت 15 سال روی این مساله مطالعه‌ی جدی‌ای انجام نشد تا $Charikar$ در سال 2000، دو نتیجه‌ی مهم در رابطه با این مساله به دست آورد: [۷]

- ارائه‌ی یک برنامه‌ریزی خطی که مساله را به صورت دقیق حل می‌کند و ارائه‌ی الگوریتمی بر پایه‌ی این برنامه

- ارائه‌ی الگوریتم 2 تقریب Greedy Peeling که زمان اجرای بسیار سریع‌تری نسبت به الگوریتم‌های دقیق پیشین داشت.

در سال 2020، Boob et al، با الهام گرفتن از الگوریتم Greedy Peeling یک نسخه‌ی تکرارشونده ازین الگوریتم ارائه کرد که به جواب بهینه‌ی مساله همگرا می‌شود که با نام Greedy++ یا Iterative Greedy Peeling شناخته می‌شود. [۴] با این حال، او نتوانست نتایج به دست آمده را به صورت تئوری اثبات کند و تنها با بررسی عملی، حدس‌هایی در رابطه با درستی الگوریتم خود زد. در سال 2022، Chekuri et al حدس Boob et al را اثبات کرد. او نه تنها برای نسخه‌ی ساده‌ی مساله، بلکه برای نسخه‌ی تعمیم‌یافته‌ی این مساله روی توابع ابرپیمانه‌ای نیز نشان داد که اجرای این الگوریتم به جواب بهینه میل می‌کند.
هم‌چنین در طول این برهه، افراد مختلفی روی حالت‌های مختلفی از این مساله که با اضافه‌کردن محدودیت‌های مختلفی به مساله به دست آمده‌اند، کار کرده‌اند که در بخش ۴ به طور دقیق‌تر این نسخه‌ها را بررسی کرده‌ایم و سعی کرده‌ایم جمع‌بندی جامعی از آخرین نتایج به دست آمده در این مسائل ارائه بدهیم. نکته‌ی قابل توجهی که در این نسخه‌ها وجود دارد این است که برخی ازین مسائل بر خلاف نسخه‌ی ساده و تعمیم آن، از نظر سختی در کلاس $\mathcal{P}$ قرار ندارند و بررسی آن‌ها ازین نظر دچار توجه ویژه‌ای است.

## ۲.۱ کارهای ما

مهم‌ترین کار ما و هسته‌ی اصلی پروژه، پیاده‌سازی الگوریتم‌ها و مطالعه‌ی این مساله از نقاط نظر مختلفی مانند زمان اجرا، ضریب تقریب و همگرایی الگوریتم‌ها است که در بخش ۵ به صورت دقیق و جزئی مورد بحث و بررسی قرار گرفته است. اما پیش از آن، به جهت آشنایی کامل و جامع با ادبیات این مساله و درک این موضوع که چه کارهایی روی این مساله انجام شده‌است و چگونه می‌توانیم تاثیری روی این مساله داشته‌باشیم، تلاش کردیم به صورت جامع الگوریتم‌های مختلف این مساله را ارائه دهیم و نتایجی که پیش‌تر به دست آمده‌اند را به صورت دقیق اثبات کنیم. هم‌چنین در اثبات‌ها، مطالبی که برای کوتاه‌شدن در سایر مقالات حذف شده‌بودند را اثبات کرده‌ایم و برخی از اثبات‌هایی که وجود داشته‌اند را با کمک مطالب کلاس، تکمیل و به بیان ساده‌تر توضیح داده‌ایم.

### ۱.۲.۱ پیاده‌سازی الگوریتم‌ها و مطالعه‌ی محاسباتی مساله

در ابتدا، ما الگوریتم‌های Goldberg، Charikar LP، Greedy Peeling و Iterative Greedy Peeling را پیاده‌سازی کردیم. در پیاده‌سازی این الگوریتم‌ها سعی شده‌است تا به نحوی پیاده شوند که از نظر زمان اجرا و هم‌چنین خوانایی الگوریتم، بهینه باشند و در نتیجه، زمان زیادی صرف فکر کردن به نحوه‌ی مناسب پیاده‌سازی این الگوریتم‌ها با داده‌ساختارهای بهینه شده‌است. کدها (به همراه نتایج و تست‌ها) را می‌توانید در گیت‌هاب پروژه مشاهده کنید.
هم‌چنین در ادامه برای تحلیل و مطالعه‌ی الگوریتم‌ها از نظر عملی و آزمایش‌کردن سوال‌های متفاوتی که در رابطه با الگوریتم‌ها به وجود می‌آید، یک دیتاست جامع طراحی کردیم. در طراحی تست‌کیس‌ها، بخشی ازین تست‌ها را مرتبط با موضوعاتی در زندگی واقعی انتخاب کرده‌ایم که این الگوریتم در آن‌ها کاربرد دارد. برای مثال، از گراف‌های نقشه‌های برخی از شهرها مثل کالیفورنیا، تگزاس و پنسیلوانیا، گراف‌های برخی شبکه‌های اجتماعی مثل فیسبوک و توییتر و برخی گراف‌های معروف دیگر استفاده کرده‌ایم. برای انتخاب این گراف‌ها، از منابع متعددی از دیتاست‌ها از جمله دیتاست استنفورد بهره برده‌ایم.
هم‌چنین برای اضافه‌کردن تست‌های تصادفی چند جنریتور مختلف برای ایجاد گراف‌های تنک و چگال و هم‌چنین جنریتوری برای ایجاد گراف‌های خاص که الگوریتم روی آن‌ها رفتار بدی دارد، پیاده‌سازی کرده‌ایم. در نتیجه، یک دیتاست شامل 77 تست‌کیس مختلف از گراف‌ها به دست آوردیم. در نهایت، موضوعات مختلفی از جمله تحلیل خروجی‌های الگوریتم‌ها و ضرایب تقریب الگوریتم‌های تقریبی، تحلیل زمان اجرای الگوریتم‌ها، بررسی ضریب تقریب الگوریتم Greedy Peeling بر حسب میزان چگال‌بودن گراف و همگرایی الگوریتم Iterative Greedy Peeling را مورد آزمایش و مطالعه قرار داده‌ایم و نتایج به دست آمده را در قالب نمودارها و جداول ارائه کرده‌ایم.
در بخش ۵ می‌توانید به صورت جزئی‌تر کارهایی که از منظر پیاده‌سازی و مطالعه‌ی محاسباتی کرده‌ایم را به همراه نتایجی که به دست آورده‌ایم و برخی مسائلی که می‌توان روی آن‌ها در آینده کار کرد را مشاهده کنید.

## ۲ الگوریتم‌های پیشین مسئله

پیش از آن که به بخش اصلی گزارش یعنی بخش 5 برسیم، بهتر است با الگوریتم‌های پیشینی که برای مسئله‌ی DSG معرفی شده‌اند، آشنا شویم و دلیل درستی آنها را ببینیم تا با ایده‌های کلاسیکی که روی این مسئله زده شده آشنا شویم و بهتر بتوانیم الگوریتم‌های جدید این مسئله را درک کنیم و نسبت به آنها شهود بیشتری داشته باشیم.
همانطور که پیش‌تر هم گفتیم مسئله چگال‌ترین زیرگراف در کلاس پیچیدگی $\mathcal{P}$ قرار دارد و برای آن الگوریتم‌های چندجمله‌ای داریم ولی از آنجایی که مسئله چگال‌ترین زیرگراف در عمل مسئله پرکاربردی هست و الگوریتم‌های دقیق آن زمان اجرای طولانی دارند، در مورد الگوریتم‌های تقریبی برای این مسئله نیز مطالعه فراوان شده برای همین این بخش را به دو قسمت تقسیم می‌کنیم و الگوریتم‌های دقیق و تقریبی این مسئله را بررسی می‌کنیم.

### ۱.۲ الگوریتم‌های دقیق

اولین الگوریتم دقیقی که برای این مسئله معرفی شد، الگوریتمی از Picard and Queyranne در [۲۰] بود که با استفاده از محاسبه‌های متوالی تعدادی جریان بیشینه، این مسئله را حل می‌کرد. بعد از گذشت چند سال Goldberg در [۱۳] زمان اجرای این الگوریتم را بهبود داد و الگوریتم Goldberg را معرفی کرد که معروف‌ترین الگوریتم دقیق این مسئله هست و توضیحات آن در این بخش آورده شده. بعدها Charikar در [۷] توانست یک توصیف دقیق از مسئله چگال‌ترین زیرگراف در غالب یک برنامه خطی بیاورد که این برنامه خطی در الگوریتم‌های دیگری که برای مسئله چگال‌ترین زیرگراف یا نسخه‌های تعمیم یافته آن داده شده‌اند اهمیت بسیار زیادی دارد، از جمله در الگوریتم Iterative Greedy Peeling که یکی از بحث‌های اصلی گزارش ماست.

#### ۱.۱.۲ الگوریتم Goldberg

در ابتدا می‌خواهیم شهودی بدهیم که اصلا ارتباط جریان بیشینه به مسئله ما چیست. فرض کنید به جای اینکه به دنبال چگال‌ترین زیرگراف باشیم، می‌خواهیم ببینیم آیا زیرگرافی هست که چگالی آن از $\lambda$ بیشتر باشد؟

$$\exists S \subseteq V \,:\, \frac{|E[S]|}{|S|} \geq \lambda \iff \exists S \subseteq V \,:\, \lambda|S| - |E[S]| \leq 0 \iff \min_{S\subseteq V}(\lambda|S| - |E[S]|) \leq 0$$

اگر بتوانیم این کار را انجام دهیم می‌توان امیدوار بود با عملی مانند جستجوی دودویی بتوانیم جواب بهینه را پیدا کنیم. با کمی دقت می‌توان دید پاسخ به روابط بالا شباهت زیادی به طراحی پیشگوی جداساز برای برنامه خطی مسئله درخت فراگیر کمینه بر اساس روش حذف گشت (Subtour Elimination) دارد و در واقعیت هم همین است و الگوریتم Goldberg در ابتدا یک کران بالا و پایین از چگالی بیشینه نگه می‌دارد و از روی گراف $G$ یک گراف جدید می‌سازد و روی آن جریان بیشینه را پیدا می‌کند تا به سوال بالا پاسخ دهد و سپس با جستجوی دودویی، مقدار چگالی بهینه را میابد.
کران پایین جواب را $\lambda_0$ و کران بالا را $\lambda_1$ بنامید. در ابتدا می‌توان قرار داد $- = \lambda_0$ و $\lambda_1 = \frac{m}{2}$ که $m$ تعداد یال‌های گراف است زیرا اگر زیرگرافی بخواهد یال داشته باشید، باید حداقل دو راس داشته باشد. حالا قرار دهید $\beta = \frac{\lambda_0+\lambda_1}{2}$ حالا بررسی می‌کنیم آیا جواب بهینه از $\beta$ بیشتر یا کمتر است. برای این گراف $G'(\beta)$ را به این صورت می‌سازیم که به $G$ دو راس $s$ و $t$ اضافه می‌کنیم. حالا از هر $s$ به $v \in V$ یالی با وزن $\frac{deg_G(v)}{2}$ می‌گذاریم و به ازای هر $uv = e \in E$ یالی به وزن $\frac{1}{2}$ از $u$ به $v$ و از $v$ به $u$ می‌گذاریم و در نهایت از هر $v \in V$ به $t$ یالی با وزن $\beta$ می‌گذاریم.

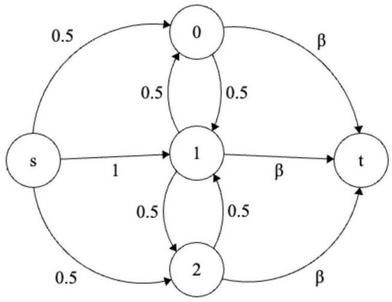
حالا وزن $s,t$ برش‌های $S$ را بررسی می‌کنیم.

**لم ۱** *برای هر $s,t$ـ برش $S$ که $s \in S$ داریم:* (*قرار دهید* $A = S - \{s\}$)

$$c(\delta(S)) = \beta|A| + m - |E[A]|$$

**برهان.** اثبات این واقعیت سرراست است و کافیست هر سه دسته یالی که در برش $S$ آمده را بنویسیم.

$$c(\delta(S)) = \sum_{v\in V-A} \frac{deg_G(v)}{2} + \sum_{e\in\delta(A,V-A)} \frac{1}{2} + \sum_{v\in A} \beta$$




شکل ۱: گراف $G'$ در حالتی که $G = P_3$ باشد.

حالا در $\sum\limits_{v\in V-A} deg_G(v)$ یال‌های $e \in \delta(A, V-A)$ یک بار می‌آیند و یال‌های $E[V-A]$ دوبار می‌آیند، پس می‌توان محاسبات را به شکل زیر ادامه داد.

$$c(\delta(S)) = |E[V-A]| + \frac{E[A, V-A]}{2} + \frac{E[A, V-A]}{2} + \beta|A| = \beta|A| + m - |E[A]|$$

■

حالا طبق لم بالا اگر در گراف جدید ساخته شده $s,t$ـ برش کمینه $S$ را بیابیم در واقع $A = S - \{s\}$ را یافتیم که مقدار $\beta|A| - |E[A]|$ را کمینه می‌کند حالا طبق محاسبات قبلی ما اگر این مقدار منفی بود یعنی چگالی چگال‌ترین زیرگراف از $\beta$ بیشتر است و می‌توانیم $\lambda_0$ را با $\beta$ جایگزین کنیم و در غیر این صورت، $\lambda_1$ را با $\beta$ جایگزین کنیم.

**لم ۲** *اگر $S_1, S_2 \subseteq V$ دو زیرگراف باشند که چگالی متفاوتی دارند، قطعا* $|d(S_1) - d(S_2)| \geq \frac{1}{n(n-1)}$

**برهان.**

$$|d(S_1) - d(S_2)| = |\frac{|E[S_1]|}{|S_1|} - \frac{|E[S_2]|}{|S_2|}| = |\frac{|E[S_1]||S_2| - |E[S_2]||S_1|}{|S_1||S_2|}|$$

حالا مسئله را دو حالت می‌کنیم:

**حالت اول** $|S_1| = |S_2| = k$ : در این حالت می‌توان عبارت را ساده‌تر کرد و به $|\frac{|E[S_1]| - |E[S_2]|}{k}|$ رسید که چون صورت آن مثبت و مخرج آن از $n$ کمتر است داریم:

$$|\frac{|E[S_1]| - |E[S_2]|}{k} \geq \frac{1}{n} \geq \frac{1}{n(n-1)}$$

**حالت دوم** $|S_1| \neq |S_2|$ در این حالت برای مخرج عبارت، کران بالای $n(n-1$ را داریم و صورت نیز مثبت است پس قطعا $|\frac{|E[S_1]||S_2| - |E[S_2]||S_1|}{|S_1||S_2|}| \geq \frac{1}{n(n-1)}$ و در این حالت نیز خواسته لم برقرار است. ■

حالا با توجه به دو لم بالا، صرفا کافی است جستجوی دودویی را تازمانی ادامه دهیم که فاصله کران پایین و بالایی که نگه داشته‌ایم، از $\frac{1}{n(n-1)}$ کمتر شود زیرا در آن لحظه می‌دانیم با اجرای یک بار دیگر الگوریتم جریان بیشینه با ثابت $\lambda_0$ برش کمینه‌ای که بدست می‌آوریم، همان چگال‌ترین زیرگراف است.

تنها مسئله باقی‌مانده، زمان اجرای این الگوریتم است.

**قضیه ۳** *الگوریتم ۱ در زمان $\mathcal{O}(T_{flow} lg\, n)$ به جواب می‌رسد. (برای مثال با استفاده از الگوریتم Edmonds-Karp زمان اجرای الگوریتم گلدبرگ $\mathcal{O}(m^2 n lg(n))$ می‌شود.)*

**برهان.** در ابتدا فاصله دو کرانی که داریم $\frac{m}{2}$ هست و در هر مرحله اجرای حلقه این فاصله نصف می‌شود پس در $\mathcal{O}(lg(mn(n-1)) = \mathcal{O}(lg(n^4)) = \mathcal{O}(lg(n))$ بار اجرای حلقه، الگوریتم به پایان می‌رسد زمان اجرای هربار حلقه نیز $T_{flow}$ هست پس به وضوح زمان اجرای الگوریتم نتیجه می‌شود. ■

**الگوریتم ۱** شبه کد الگوریتم گلدبرگ

**ورودی:** گراف ساده‌ی $G = (V, E)$.
**خروجی:** $S \subseteq V$ به طوری که $G[S]$ بیشترین چگالی را داشته باشد.
(۱) $\lambda_0 \leftarrow 0$، $\lambda_1 \leftarrow \frac{m}{2}$
(۲) تا زمانی که $\lambda_1 - \lambda_0 \geq \frac{1}{n(n-1)}$ هست:
(۳) $\beta \leftarrow \frac{\lambda_1+\lambda_0}{2}$
(۴) $s, t$ـبرش کمینه $S$ را روی گراف $G'(\beta)$ پیدا کن.
(۵) $A \leftarrow S - \{s\}$
(۶) اگر $A$ مجموعه تهی بود:
(۷) $\lambda_1 \leftarrow \beta$
(۸) در غیر این صورت:
(۹) $\lambda_0 \leftarrow \beta$
(۱۰) $s, t$ـبرش کمینه $S$ را روی گراف $G'(\lambda_0)$ پیدا کن.
(۱۱) $S - \{s\}$ را خروجی بده.

### ۲.۱.۲ برنامه خطی Charikar

حالا الگوریتم برپایه برنامه خطی Charikar را معرفی می‌کنیم. او در [۷] نشان داد جواب بهینه این برنامه خطی همیشه برابر با چگالی چگال‌ترین زیرگراف است. در واقع اگر به طراحی یک برنامه خطی برای این مسئله فکر کنید، ایده اولیه این است که برای هر راس یک متغییر $y_v$ بذاریم که یک بودن این متغیر نشان دهنده آمدن این راس در زیرگراف بهینه است. همچنین مشابها باید برای هر یال یک متغییر $x_e$ بذاریم. چالشی که اینجا با آن مواجه می‌شویم طراحی تابع هدف است. درواقع وجود $|S|$ در مخرج باعث می‌شود این ایده به نتیجه نرسد. Charikar برای حل این مشکل از نوعی تنکیک Load Balancing استفاده کرد. در واقع او سعی کرد طوری قیدها را قرار دهد که زمانی که یک یال در زیرگراف آمده باشد، $x_e = \frac{1}{|S|}$ بشود. واضح است اگر بتوانیم اینکار را بکنیم تابع هدف ما تابع بسیار ساده $\sum_{e \in E} x_e$ می‌شود. برنامه خطی ارائه شده توسط Charikar برنامه زیر است.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} x_e \\
\text{to subject} \quad & x_{u,v} \leq y_u \ \forall uv \in E \\
& x_{u,v} \leq y_v \ \forall uv \in E \\
& \sum_{v \in V} y_v = 1 \\
& x, y \geq 0
\end{aligned}
\tag{۱}
$$

حال خصوصیات این برنامه خطی را بررسی می‌کنیم.

**لم ۴** *برای هر زیرگراف $S$ با چگالی $d$، جوابی متناظر با $S$ در برنامه خطی ۱ داریم که حاصل تابع هدف آن $d$ است.*

**برهان.** واضح است با قرار دادن مقدار $y_v = \frac{1}{|S|}$ برای راس‌های $v \in S$ و قرار دادن $x_e = \frac{1}{|S|}$ برای یال‌های $e \in E$، به خواسته قضیه می‌رسیم. ■

حالا الگوریتم را بیان می‌کنیم. ایده این الگوریتم شباهت زیادی به تکنیک‌های گردکردنی که برای مسئله‌ی درخت آشتانیر جایزه جمع‌کن در کلاس درس دیدیم، دارد.

واضح است در خط دوم الگوریتم صرفا کافی است برای متناهی مقدار ممکن(همان مقادیر $y_v$ها ) مقدار $S_r$ را بیابیم و فقط برای تحلیل راحت‌تر، الگوریتم را اینگونه نوشتیم. حالا برای تحلیل الگوریتم از استراتژی مشابه استراتژی‌هایی که در درس برای مسئله‌های چند برش یا درخت اشتاینر جایزه جمع کن دیدیم، استفاده می‌کنیم.

**الگوریتم ۲** شبه کد الگوریتم چاریکار

**ورودی:** گراف ساده‌ی $G = (V, E)$.
**خروجی:** $S \subseteq V$ به طوری که $G[S]$ بیشترین چگالی را داشته باشد.
(۱) برنامه خطی ۱ را حل کن و جواب بهینه $x^*, y^*$ را بیابید.
(۲) برای هر $0 \leq r \leq y^*_{max}$ قرار بده $S_r = \{v \,:\, y_v \geq r\}$
(۳) $S_r$ ای که بیشترین چگالی را دارد خروجی بده.

**قضیه ۵** *الگوریتم بیان شده در ۲ یک الگوریتم دقیق برای مسئله‌ی چگال‌ترین زیرگراف است.*

**برهان.** فرض کنید برای هر $r$ داشته باشیم $\frac{|E[S_r]|}{|S_r|} < OPT$ آنگاه می‌توان گفت $|E[S_r]| < |S_r|OPT$ پس محاسبات زیر را داریم:

$$\int_0^{y^*_{max}} |E[S_r]|dr < OPT \int_0^{y^*_{max}} |S_r|dr)$$

حالا دو طرف نامساوی را محاسبه می‌کنیم.

$$\int_0^{y^*_{max}} |E[S_r]|dr = \sum_{e\in E} \int_0^{y^*_{max}} \mathbb{1}(e \in S_r)dr = \sum_{e\in E} min(y^*_u, y^*_v) = \sum_{e\in E} x_e = OPT$$

همچنین مشابها برای طرف دیگر نامساوی داریم:

$$\int_0^{y^*_{max}} |S_r|dr) = \sum_{v\in V} \int_0^{y^*_{max}} \mathbb{1}(v \in S_r)dr = \sum_{v\in V} y_v = 1$$

حالا با کنار هم گذاشتن معادلات بالا به این تناقض می‌رسیم که $OPT < OPT$ پس قطعا یک $r$ داریم که $S_r$ چگالی بیشتر مساوی از $OPT$ دارد حالا چون طبق لم۴ $OPT \geq d(S_r)$ پس $d(S_r) = OPT$ و خواسته اثبات شد. ■

## ۲.۲ الگوریتم‌های تقریبی

همانطور که قبل‌تر بیان کردیم با وجود داشتن الگوریتم چندجمله‌ای برای این مسئله، در مورد الگوریتم‌های تقریبی آن نیز مطالعه فراوان شده است. اولین الگوریتم تقریبی برای این مسئله، توسط Kortsarz and Peleg در [۱۷] ارائه شد. آنها نشان دادند $w$ـ هسته بیشینه در گراف، یک $2$ـ تقریب برای مسئله ماست.

**تعریف ۱** *به زیرگراف ماکسیمالی که درجه‌ی رئوسش حداقل $w$ باشد، $w$ـ هسته می‌گویند.*

ممکن است برایتان سوال شود که ارتباط این مفهوم $w$ـ هسته با مسئله چگال‌ترین زیرگراف چیست. در واقع با کمی دقت واضح است چگالی یک زیرگراف، چیزی جز نصف متوسط درجه راس‌های آن زیرگراف نیست، به همین دلیل می‌توان برای پیدا کردن زیرگراف با درجه متوسط بالا به دنبال زیرگراف با درجه‌های بالا گشت و این همبستگی بین دو مسئله وجود دارد. در ادامه Charikar در [۷] با الهام از همین ایده یک الگوریتم حریصانه‌ی $2$ـ تقریب با زمان اجرای خطی برای این مسئله طراحی کرد که معروف‌ترین الگوریتم تقریبی این مسئله است. در بخش۵ این الگوریتم را روی تست‌های مختلف بررسی کردیم و نشان دادیم ضریب تقریب آن در عمل خیلی بهتر از $2$ هست. پس از این الگوریتم Boob et al در [۴] با ایده اجراهای متوالی الگوریتم حریصانه الگوریتم جدیدی طراحی کردند و حدس زدند که این الگوریتم به جواب بهینه میل می‌کند. Chekuri, Quanrud, Torres در [۸] این حدس را ثابت کردند که این‌ها را در ۳ بررسی کرده‌ایم.

### ۱.۲.۲ الگوریتم Gready Peeling

فرض کنید می‌خواهد به طور حریصانه چگالی یک زیرگراف را زیاد کنید. اگر یک راس از این زیرگراف حذف کنید از صورت کسر چگالی به اندازه درجه این راس و از مخرج کسر چگالی یک واحد کم کرده‌اید. پس به طور حریصانه بهترین انتخاب

برای حذف کردن از زیرگراف حذف کردن راسی است که کم‌ترین درجه را دارد. این شهود کلی الگوریتمی است که در اینجا می‌خواهیم معرفی کنیم.

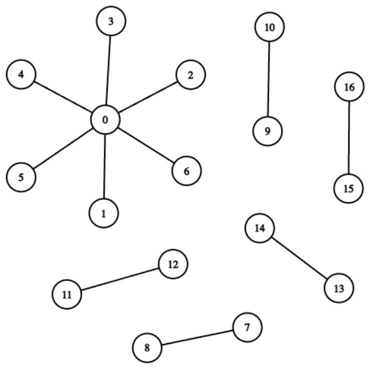
**الگوریتم ۳** شبه کد الگوریتم Greedy Peeling

**ورودی:** گراف ساده‌ی $G = (V, E)$.
**خروجی:** $S \subseteq V$ به طوری که $G[S]$ بیشترین چگالی را داشته باشد.
(۱) $S_n \leftarrow V$
(۲) برای $i$ از $n$ تا $1$ انجام بده:
(۳) راس با درجه کمینه در $G[S_i]$ را پیدا کن و آن را $v$ بنام.
(۴) $S_{i-1} \leftarrow S_i - \{v\}$
(۵) $j \leftarrow argmax_i \ density(S_i)$
(۶) $S_j$ را خروجی بده.

البته این الگوریتم ارتباط زیادی با تعریف $w$ــ هسته هم دارد در واقع قضیه زیر را داریم.

**قضیه ۶** *الگوریتم Greedy Peeling همه‌ی $w$ــ هسته‌های گراف را پیدا می‌کند.*

از آنجایی که نسخه کمی متفاوتی ازین الگوریتم را در ۳ هم می‌آوریم از آوردن اثبات درستی و ضریب تقریب آن در اینجا صرف نظر شده.
با توجه به شهودهایی که از این الگوریتم دادیم حس می‌شود این الگوریتم روی حالت‌هایی که در چگال‌ترین زیرگراف راس‌هایی با درجه خیلی کم باشند بد کار می‌کند در واقعیت هم همین است و در اینجا با آوردن مثال‌هایی الهام گرفته شده از [۱۴] نشان می‌دهیم ضریب تقریب 2 این الگوریتم، دقیق است.




شکل ۲: یک مثال از نمونه بد برای الگوریتم در حالتی که $p = 6$ و $t = 5$ باشد.

**قضیه ۷** *ضریب تقریب 2 برای الگوریتم Greedy Peeling ، دقیق است.*

**برهان.** مثال را اینگونه می‌سازیم. در ابتدا $2t$ راس مختلف قرار دهید و بین آنها یک تطابق بذارید. در واقع $t$ یال کاملا ایزوله بسازید. حالا $p+1$ راس اضافه کنید و یکی از رئوس را به $p$ راس دیگر وصل کنید. (یک ستاره‌ی $p+1$ راسی) حالا به راحتی می‌توان دید چگال‌ترین زیرگراف این گراف همان ستاره‌ی $p+1$ راسی است که چکالی آن $\frac{p-1}{p}$ است. حالا فرض کنید الگوریتم بین چند راسی برای حذف در صورت وجود همیشه راسی از ستاره را حذف کند. در این حالت الگوریتم در هر مرحله یک راس را از ستاره حذف می‌کند تا در نهایت کل ستاره حذف می‌شود و فقط $t$ یال ایزوله باقی می‌ماند. و سپس از بین آنها راس حذف می‌کند می‌توان دید که در این حالت با حذف هر راس چگالی کاهش پیدا می‌کند پس چگالی بیشینه

که الگوریتم می‌بیند و خروجی می‌دهد همان چگالی اولیه گراف است. حالا گراف ما $2t+p+1$ راس داشت و $t+p$ یال داشت پس خروجی الگوریتم $\frac{t+p}{2t+p+1}$ است.

$$Algorithms\ Answer : \frac{t+p}{2t+p+1}\ ,\ Optimal\ Answer : \frac{p-1}{p}\ \rightarrow\ Aprroximate\ Ratio = \frac{\frac{t+p}{2t+p+1}}{\frac{p-1}{p}}$$

حالا اگر $p$ به بی‌نهایت میل کند و $t << p$ باشد عبارت بالا، به $\frac{1}{2}$ میل می‌کند. ■

در مورد زمان اجرای این الگوریتم و نحوه پیاده‌سازی آن به طور دقیق در بخش۵ صحبت کرده‌ایم.

# ۳ الگوریتم Iterative Greedy Peeling

در این بخش قصد داریم الگوریتم Iterative Greedy Peeling را معرفی کنیم که به آن Greedy++ هم می‌گویند. ایده اولیه این الگوریتم توسط Boob et al در [۴] مطرح شد ولی آنها نتوانستند در مورد کارایی الگوریتم و ضریب تقریب آن چیزی بگویند تا اینکه به تازگی Chekuri, Quanrud, Torres در [۸] توانستند نشان دهند این الگوریتم به جواب بهینه مسئله چگال‌ترین زیرگراف میل می‌کند. ایده اولیه این الگوریتم برخواسته از روش Multipicative Weight Update هست و در کل ایده طبیعی دارد. درواقع از آنجایی که روش Greedy Peeling خوب کار می‌کند و معمولا در عمل بسیار بهتر از ضریب تئوری خودش عمل می‌کند این ایده که بارهای متوالی الگوریتم Greedy Peeling را اجرا کنیم و بعد از هر بار اجرا برای راس‌های گراف یک معیاری از ارزشمندی آن راس بدست بیاوریم که در نوبت‌های بعدی اجرای Greedy Peeling به جای حذف راس‌هایی که کم‌ترین درجه را دارند راس‌هایی که طبق معیار ما کمترین ارزش را دارند حذف کنیم.
ما در این بخش در ابتدا کمی در مورد الگوریتم Greedy Peeling و ویژگی‌های آن صحبت می‌کنیم سپس به سراغ الگوریتم Greedy++ می‌رویم و تلاش می‌کنیم توضیح دهیم چرا این الگوریتم خوب عمل می‌کند و می‌تواند برای هر $\epsilon > 0$ به ما یک $1-\epsilon$ تقریب بدهد.

## ۱.۳ الگوریتم Greedy Peeling برای مسئله‌ی چگال‌ترین زیرگراف ابرپیمانه‌ای

همانطور که از نام این بخش معلوم است قراراست به‌جای اینکه تمرکزمان به مسئله‌ی چگال‌ترین زیرگراف عادی باشد به نسخه جامع‌تر آن یعنی چگال‌ترین زیرگراف ابرپیمانه‌ای نگاه کنیم که به جای یافتن زیرگراف $S \subseteq V$ که $\frac{|E[S]|}{|S|}$ می‌خواهیم زیرگرافی بیابیم که $\frac{f(S)}{|S|}$ را بیشینه کند که $f$ یک تابع ابرپیمانه‌ای نامنفی یکنوا دلخواه است که در آن داریم $f(\emptyset) = 0$ نه لزوما تابع $|E[S]|$ (در کلاس درس دیدیم که تابع $|E[S]|$ ابرپیمانه‌ای است). در این بخش از این به بعد وقتی در مورد تابع $f$ صحبت می‌کنیم فرض می‌کنیم شرایط بالا را دارد.
برای آنکه توضیحات را بیان کنیم ابتدا چند تعریف را بیان می‌کنیم.

**تعریف ۲** *برای تابع $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ و زیرمجموعه $S \subseteq V$ و $v \in V - S$ تعریف می‌کنیم:*

$$f(v|S) = f(S) - f(S - \{v\})$$

**تعریف ۳** *یک تعریف معادل با تعریفی که در کلاس داشتیم برای تابع ابرپیمانه‌ای $f$ این است که با ازای هر $A \subset B \subset V$ و $x \in V - B$ داشته باشیم:*

$$f(x|A) \geq f(x|B)$$

**تعریف ۴** *برای هر تابع $f$ پارامتر $c_f$ را اینگونه تعریف می‌کنیم.*

$$c_f = \max_{S \subseteq V} \frac{\sum\limits_{v \in s} f(v|S-v)}{f(S)}$$

**قضیه ۸** *در حالت $f(S) = |E[S]|$ مقدار $c_f = 2$ است.*

**برهان.** به سادگی می‌توان دید اگر $f(S)=|E[S]|$ باشد، $f(v|S-v)$ درجه $v$ در $G[S]$ می‌شود و چون جمع درجه راس‌های یک گراف دوبرابر تعداد یال‌های آن هست پس در این حالت $c_f = 2$ خواهد بود. ■

**لم ۹** *برای هر تابع $f$ و $S \subseteq V$ داریم:*

$$\sum_{v\in s} f(v|S-v) \geq f(S)$$

**برهان.**

$$\sum_{v\in s} f(v|S-v) = \sum_{v\in s}(f(S)-f(S-v)) = |S|f(S) - \sum_{v\in s} f(S-v)$$

حالا از خاصیت $f(A\cup B)+f(A\cap B) \geq f(A)+f(B)$ در توابع ابرپیمانه‌ای استفاده می‌کنیم.
فرض کنید $S=\{v_1,\dots v_{|S|}\}$ باشد.

$$f(S-v_1)+\cdots+f(S-v_{|S|}) \leq f(S)+f(S-\{v_1,v_2\}+f(S-v_3)+\cdots+f(S-v_{|S|})$$

$$\leq \cdots \leq (|S|-1)f(S)+f(\emptyset) = (|S|-1)f(S)$$

حالا با کنار هم گذاشتن محاسبات بالا به این می‌رسیم که

$$\sum_{v\in s} f(v|S-v) \geq |S|f(S) - (|S|-1)f(S) = f(S)$$

و لم اثبات شد. ■

حالا می‌توانیم تحلیل الگوریتم را ادامه دهیم ممکن است برایتان سوال شود الان که $f$ یک تابع دلخواه است دیگر در الگوریتم Greedy Peeling چطوری راس‌ها را حذف می‌کنیم زیرا دیگر منطقی نیست که بر اساس درجه راس‌ها اینکار را بکنیم برای همین در مسئله‌ی چگال‌ترین زیرگراف ابرپیمانه‌ای الگوریتم Greedy Peeling در هر لحظه از مجموعه $Si$ از راس‌های کنونی راس $v$ را حذف می‌کند که $f(v|S)$ برای آن کمینه باشد و $S_{i-1}=S_i-v$ قرار می‌دهد.

**لم ۱۰** *اگر $S^*$ جواب بهینه مسئله باشد و $d^* = \frac{f(S^*)}{|S^*|}$ باشد آنگاه به ازای هر $v\in S^*$ داریم $f(v|S^*) \geq d^*$ است.*

**برهان.** اگر $S^*$ تک عضوی باشد حکم بدیهی است زیرا $f(v|S^*) = f(S^*)-f(\emptyset) = f(S^*) = d^*$ پس فرض کنید $S^*$ بالای یک عضو دارد. بنا به برهان خلف اگر قضیه برقرار نباشد یک $v\in S^*$ هست که $f(v|S^*) < d^*$ است. حالا چگالی $S^*-v$ را در نظر بگیرید.

$$\frac{f(S^*-v)}{|S^*-v|} > \frac{f(S^*)-d^*}{|S^*|-1} = d^*$$

ولی این با بهینه بودن $S^*$ تناقض دارد و لم با استفاده از برهان خلف ثابت شد. ■

**قضیه ۱۱** *الگوریتم Greedy Peeling برای یافتن چگال‌ترین زیرگراف ابرپیمانه‌ای یک الگوریتم $\frac{1}{c_f}$ـ تقریب است.*

**برهان.** جواب بهینه مسئله را $S^*$ بگیرید و قرار دهید $d^* = \frac{f(S^*)}{|S^*|}$. حالا فرض کنید $S_i$ مجموعه مینیمالی در فرایند اجرای الگوریتم Greedy Peeling باشد که $S^* \subseteq S_i$ پس باید در مرحله $i$ ام الگوریتم از $S_i$ یک $v_j \in S^*$ را حذف کرده باشد یا معادلا $v_j = argmin_{v_k\in S_i} f(v_k|S_i)$ را داشته باشیم. حالا چگالی $S_i$ را حساب می‌کنیم.

$$\frac{f(S_i)}{|S_i|} = \frac{f(S_i)}{|S_i|} = \frac{\sum\limits_{v\in s} f(v|S_i-v)}{|S_i|} \times \frac{f(S_i)}{\sum\limits_{v\in s} f(v|S_i-v)}$$

حالا با استفاده از $v_j = argmin_{v_k \in S_i} f(v_k|S_i)$ و تعریف پارامتر $c_f$ داریم:

$$\frac{f(S_i)}{|S_i|} \geq \frac{1}{c_f} \times \frac{|S_i| f(v_k|S_i)}{|S_i|} = \frac{f(v_k|S_i)}{c_f}$$

حالا چون $S^* \subset S_i$ طبق تعریف۳ داریم $f(v_k|S_i) \geq f(v_k|S^*)$ و همچنین در لم۱۰ دیدیم $f(v_k|S^*) \geq d^*$ پس محاسبات بالا را می‌توان به شکل زیر ادامه داد.

$$\frac{f(S_i)}{|S_i|} \geq \frac{f(v_k|S_i)}{c_f} \geq \frac{f(v_k|S^*)}{c_f} \geq \frac{d^*}{c_f}$$

پس خود $S_i$ یک $\frac{1}{c_f}$ تقریب است و چون الگوریتم از میان همه $S_k$ها بهترین را خروجی می‌داد الگوریتم نیز $\frac{1}{c_f}$ هست. ■

یک نتیجه‌ی مهم از قضیه۱۱ این است که الگوریتم Greedy Peeling برای مسئله چگال‌ترین زیرگراف عادی 2ـ تقریب است.

حالا که قضیه۱۱ را بیان کردیم تنها چالشی که باقی می‌مان یافتن مقدار دقیق یا کران بالای $c_f$ هست که باید با توجه به خود تابع $f$ و ویژگی‌هایی که دارد آن را پیدا کنیم، مثل استدلالی که برای تابع $f(S) = |E[S]|$ کردیم.

## ۲.۳ الگوریتم Iterative Greedy Peeling

مطابق با بحث‌های انجام شده در قبل، این الگوریتم باید به طریقی از اجراهای قبلی الگوریتم Greedy Peeling استفاده کند و یک وزن براساس مهم بودن هر راس در حل مسئله به آن راس نسبت دهد.

یک انتخاب طبیعی برای نسبت دادن وزن به راس‌ها این است که برای هر راس یک مقدار $l_v$ نگه داریم که در ابتدا مقدار آن صفر است و اگر در مرحله‌ای از اجرای الگوریتم راس $v$ از یک $S_i$ حذف شد مقدار $l_v$ را به شکل زیر تغییر دهیم $l_v = l_v + deg_{G[S_i]}(v)$ آنگاه این الگوریتم از درجه یک راس در دفعات قبلی اجرای الگوریتم Greedy Peeling به عنوان یک راهنما برای تاثیرگذاری این راس در پیدا کردن چگال‌ترین زیرگراف استفاده می‌کند.

با توجه به توصیفات بالا شبه کد الگوریتم به شکل زیر خواهد شد:

---

**الگوریتم ۴** شبه کد الگوریتم Iterative Greedy Peeling

---

**ورودی:** گراف ساده‌ی $G = (V, E)$ و عدد صحیح مثبت $T$.

**خروجی:** $S \subseteq V$ به طوری که $G[S]$ بیشترین چگالی را داشته باشد.

(۱) برای هر $i$ از $1$ تا $T$ انجام بده:

(۲) $S_{i,n} \leftarrow V$

(۳) برای $j$ از $n$ تا $1$ انجام بده:

(۴) راس $v$ که برای آن مقدار $l_v + deg_{G[S_{i,j}]}(v)$ کمینه است را پیدا کن.

(۵) $S_{i,j-1} \leftarrow S_{i,j} - \{v\}$

(۶) $l_v \leftarrow l_v + deg_{G[S_{i,j}]}(v)$

(۷) $i, j \leftarrow argmax_{i,j}\ density(S_{i,j})$

(۸) $S_{i,j}$ را خروجی بده.

---

الگوریتم بالا برای اولین در [۴] مطرح شد ولی اثباتی مبتنی بر ضریب تقریب آن ارائه نشد تا اینکه به تازگی Chekuri, Quanrud, Torres در [۸] قضیه زیر را در مورد این الگوریتم اثبات کردند.

**قضیه ۱۲** *برای گراف $G$ اگر $\Delta$ ماکسیمم درجه $G$ و $d^*$ چگالی بیشینه در $G$ باشد آنگاه برای $T \geq \mathcal{O}(\frac{\Delta ln(n)}{d^* \epsilon})$ الگوریتم۴ یک خروجی $1-\epsilon$ تقریب برای مسئله‌ی چگال‌ترین زیرگراف می‌دهد.*

در واقع نتیجه‌ای که در [۸] نشان داده شده کمی از قضیه بالا قوی‌تر است و برای مسئله‌ی چگال‌ترین زیرگراف ابرپیمانه‌ای داده شده است.

# ۴ نسخه‌های مختلف مسئله چگال‌ترین زیرگراف

نسخه‌های مختلف مسئله چگال‌ترین زیرگراف را در حالت کلی می‌توان به دو دسته تقسیم کرد.

۱. مسائل دارای قید روی اندازه زیرگراف

۲. مسائل دارای قید روی همبندی زیرگراف

در این بخش ما این دو دسته از مسائل را معرفی و آنها را از لحاظ سختی محاسباتی بررسی و به بعضی الگوریتم‌های مطرح برای آنها اشاره می‌کنیم.
هدف اصلی گزارش ما بررسی خود مسئله‌ی چگال‌ترین زیرگراف است و این قسمت به منظور کامل بودن گزارش آمده و خلاصه‌تر نوشته شده است. به همین خاطر بیشتر تلاش شده تا به منابع مفید برای مطالعه اضافه این بخش ارجاع داده شود.

## ۱.۴ مسائل دارای قید روی اندازه زیرگراف

از جمله‌این مسائل می‌توان به مسائل DKSG و DALKSG و DAMKSG اشاره کرد که به ترتیب می‌گوید اندازه زیرگراف انتخاب شده باید دقیقا، حداقل و یا حداکثر $K$ باشد. در ادامه این سه مسئله را بررسی می‌کنیم.

### ۱.۱.۴ مسئله‌ی Densest k Subgraph

همانطور که بالاتر گفتیم این مسئله به دنبال یافتن چگال‌ترین زیرگراف دقیقا $k$ راسی است یعنی $\max_{S\subseteq V,|S|=k} \frac{|E[S]|}{|S|}$ واضح است که مخرج این کسر همواره برابر با $k$ است و تاثیری در نتیجه بهینه‌سازی ما ندارد پس هدف ما یافتن زیرگراف $k$ راسی است که بیشترین تعداد یال را دارد. زیرگرافی که بیشترین تعداد یال را می‌تواند داشته باشد یک خوشه هست پس به نظر می‌رسد میان این مسئله و مسئله یافتن یک خوشه در گراف ارتباطاتی هست و در واقع هم همین هست و می‌توان نشان داد $Clique \leq_P DKSG$ و از آنجایی که $Clique$ یک مسئله $\mathcal{NP}$-Hardهست این مسئله نیز $\mathcal{NP}$-Hardمی‌شود.

**قضیه ۱۳** $Clique \leq_P DKSG$

**برهان.** واضح است در صورت وجود یک خوشه $k$ راسی در گراف آنگاه چگال‌ترین زیرگراف $k$ راسی این گراف همین خوشه خواهد بود و چگالی آن $\frac{k-1}{2}$ است. پس می‌توان برای هر $1 < r \leq n$ چگال‌ترین زیرگراف $r$ راسی را پیدا کرد. حالا بزرگ‌ترین $r$ که برای آن زیرگراف $r$ راسی با چگالی $\frac{r-1}{2}$ داشتیم اندازه بزرگ‌ترین خوشه گراف خواهد بود و به این ترتیب کاهش کامل می‌شود. ■

الگوریتم‌های تقریبی چندجمله‌ای زیادی برای این مسئله ارائه شده‌اند برای مثال الگوریتم $\mathcal{O}(n^{\frac{1}{3}})$ـ تقریب Feige, Kortsarz, and Peleg [۱۱] که مشابه با تکنیکی که در مسئله‌ی Max Sat در کلاس دیدیم که با ترکیب دو روش مختلف به روش بهتری رسیدیم، تلاش کرد با حساب کردن سه کاندیدا به روش‌های مختلف و خروجی دادن بهترین زیرگراف از بین اینها مسئله را حل کند که در میان این روش‌ها یک روش به طور تصادفی زیرگراف را انتخاب می‌کرد که در $k$ های بزرگ خوب عمل می‌کرد و یک روش دیگر نیز به صورت حریصانه گراف را می‌ساخت.
همچنین علاوه بر پیچیدگی محاسباتی این مسئله تحقیقاتی نیز در مورد پیچیدگی پارامترایزد این مسئله هم انجام شده مانند قضیه زیر از Cai L که در [۶] آمده.

**قضیه ۱۴** *مسئله $DKSG$ در کلاس W[1]-Hard قرار دارد.*

**برهان.** در ابتدا دو نکته را بیان می‌کنیم.
**نکته اول** اگر یک زیرگراف $k$ راسی $G[S]$ تعداد $E[S]$ یال داشته باشد آنگاه زیرگراف $G^c[S]$ به وضوح $\binom{k}{2} - E[S]$ یال دارد پس چگال‌ترین زیرگراف $k$ راسی در $G$ همان کم‌چگال‌ترین زیرگراف $k$ راسی در $G^c$ است.
**نکته دوم** فرض کنید گراف $G$ گرافی $r$ـ منظم باشد در این حالت اگر یک زیرگراف $G[S]$ در گراف $G$ تعداد $E[S]$ یال داشته باشد آنگاه هر راس این موجوعه $r$ یال را می‌پوشاند و یال‌های $E[S]$ دوبار شمارده می‌شوند پس $S$ تعداد $r|S| - E[S]$ یال را می‌پوشاند پس به راحتی می‌توان دید پوشش کمینه $k$ راسی در $G$ همان چگال‌ترین زیرگراف $k$ راسی خواهد بود.
**اثبات** حالا با کنار هم گذاشتن این دو نکته می‌فهمیم در گراف‌های منظم مسئله‌های چگال‌ترین زیرگراف $k$ راسی و پوشش $k$ راسی بیشینه هم‌ارزند. حالا چون مسئله پوشش $k$ راسی بیشینه در گراف‌های منظم W[1]-Hard هست مسئله چگال‌ترین

زیرگراف $k$ راسی در گراف‌های منظم هم W[1]-Hard می‌شود پس، مسئله کلی‌تر چگال‌ترین زیرگراف در هرگراف دلخواه $G$ نیز W[1]-Hard هست. ■

همچنین این مسئله در حالت‌هایی که روی مقدار $k$ یا تعداد یال‌های گراف شرط اضافه‌ای بذاریم نیز بررسی شده است.

**قضیه ۱۵** *در حالت $k = \frac{n}{2}$ الگوریتم بدیهی $\frac{1}{4}$ـ تقریب داریم.*

**برهان.** اگر به طور کاملا یکنواخت زیرمجموعه $\frac{n}{2}$ راسی $S$ را انتخاب کنیم احتمال انتخاب شدن هر راس $\frac{1}{2}$ خواهد بود پس داریم:

$$P[v \in s] = \frac{1}{2} \to p[uv = e \in S] = P[u \in s]P[v \in s] = \frac{1}{4} \to \mathbb{E}[|E[S]|] = \frac{m}{4}$$

حالا با روش‌های تصادف‌زدایی که در کلاس هم دیده بودیم می‌توان یک زیرمجموعه $S$ پیدا کرد که تعداد یال‌های $E[S]$ از $\frac{m}{4}$ بیشتر است. چون بیشترین تعداد یال ممکن که یک زیرگراف می‌تواند داشته باشد $m$ هست پس این یک $\frac{1}{4}$ـ تقریب برای مسئله‌ی ما است. ■

اما این بهترین نتیجه ممکن نیست و Ye, Zhang در [۲۲] با استفاده از ترکیب برنامه‌ریزی نیمه معین و الگوریتم‌های گردکردن به یک الگوریتم $0.586$ـ تقریب برای این مسئله در این حالت خاص رسیدند و الگوریتم‌های تقریبی قبلی را بهبود دادند. همچنین آنها در این پیپر ارتباط مسئله DKSG در حالت $k = \frac{n}{2}$ را با مسئله Min-Bisection که در کلاس درس نیز دیده بودیم بررسی می‌کنند.

### ۲.۱.۴ مسئله‌ی Densest At Least k Subgraph

این مسئله از لحاظ پیچیدگی محاسباتی در کلاس $\mathcal{NP}$-Hard قرار دارد ولی اثبات این موضوع به سادگی اثباتی پیچیدی مسائل دیگری که در این بخش گفتیم نیست. Khuller and Saha در [۱۶] با کاهش دادن مسئله‌ی DKSG به این مسئله این واقعیت را نشان دادند.

**قضیه ۱۶** *مسئله‌ی $DALKSG$ یک مسئله‌ی $\mathcal{NP}$-Hard هست و $DALKS \leq_p DKSG$*

**برهان.** فرض کنید به ما گراف $n$ راسی $G$ و عدد صحیح مثبت $k$ و عدد $d$ داده شده و می‌خواهیم بدانیم آیا زیرگراف $k$ راسی در $G$ وجود دارد که چگالی آن بیشتر از $d$ باشد.
حالا به گراف $G$ یک گراف کامل $n^2$ راسی اضافه کنید تا به گراف $G'$ برسیم. درواقع $G'$ اجتماع $G$ و یک $K_{n^2}$ است. حالا روی این گراف $G'$ چگال‌ترین زیرگراف حداقل $n^2 + k$ راسی را پیدا می‌کنیم و آن را $S$ می‌نامیم. ادعا می‌کنیم این جواب چند ویژگی خوب دارد.
**ویژگی ۱:** کل راس‌های $K_{n^2}$ در $S$ آمده.
**اثبات:** فرض کنید چنین نباشد. مجموعه راس‌هایی از این گراف کامل که در $S$ نیامده $T$ می‌نامیم. همچنین فرض کنید $|S| = r$ و چگالی $S$ را با $d(S)$ نشان دهید. حالا حساب می‌کنیم در صورت اضافه کردن $T$ به $S$ تعداد یال‌های آمده در زیرگراف چقدر زیاد می‌شود. هر راس در $T$ به $n^2$ راس وصل است ولی یال‌های میان خود $T$ دوبار شمارده‌ایم پس تعداد یال‌هایی که اضافه می‌شود $|E_T| = |T|n^2 - \frac{|T|(|T|-1)}{2}$ هست. حالا نامساوی زیر را نشان می‌دهیم.

$$|E_T| = |T|n^2 - \frac{|T|(|T|-1)}{2} = |T|(n^2 - \frac{|T|-1}{2}) \geq |T|(\frac{n^2-1}{2}) \geq |T|d(S)$$

که دلیل نامساوی آخر این است که زمانی که $T$ ناتهی باشد در $S$ هر دو بخش $S - K_{n^2}$ و $K_{n^2} - T$ چگالی کمتر از $\frac{n^2-1}{2}$ دارد.
پس برای چگالی جواب جدید خواهیم داشت:

$$d(S \cup T) = \frac{d(S)r + |E_T|}{r + |T|} > \frac{d(S)(r + |T|)}{r + |T|} = d(S)$$

و ویژگی اول اثبات می‌شود.
**ویژگی ۲:** دقیقا $k$ راس از $G$ در $S$ آمده، یعنی $|S \cap G| = k$ است.

**اثبات:** دلیل این واقعیت این است که $d(S)$ نسبت به $|S \cap G|$ نزولی است و ترجیح می‌دهد کمترین تعداد ممکن راس از $G$ بردارد ولی چون باید حداقل $n^2+k$ راس بردارد مجبور است دقیقا $k$ راس از $G$ بردارد. فرض کنید در صورتی که از $G$ حتما $l$ راس برداشته شده باشد جواب بهینه $S_l$ باشد حالا دلیل نزولی بودن به شکل زیر است:

$$d(S_l) \leq \frac{\binom{n^2}{2}+\binom{l}{2}}{n^2+l} \leq \frac{\binom{n^2}{2}+1}{n^2+l-1} \leq d(S_{l-1})$$

و این ویژگی نیز اثبات شد.
حالا با کنار هم گذاشتن دو ویژگی بالا مشخص می‌شود که $k$ راسی که از $G$ در $S$ آمده‌اند دقیقا چگال‌ترین زیرگراف $k$ راسی $G$ را تشکیل می‌دهند و چگالی این زیرگراف برابر با: $\frac{d(S)(n^2+k)-\binom{n^2}{2}}{k}$ هست. پس اگر $d(S) > \frac{dk+\binom{n^2}{2}}{n^2+k}$ باشد آنگاه چگال‌ترین زیرگراف $k$ راسی $G$ چگالی بیشتر از $d$ دارد. ■

در مورد الگوریتم‌هایی که برای این مسئله وجود دارد Andersen and Chellapilla در [۱] نشان دادند که الگوریتم Greedy Peeling که برای مسئله DSG معرفی کردیم به ما یک $3$ـ تقریب در زمان خطی می‌دهد که اثبات این ضریب مانند اثبات $2$ـ تقریب بودن الگوریتم Greedy Peeling است و از خاصیت‌های مربوط به $w$ـ هسته گراف استفاده می‌کند که در تعریف۱ معرفی کردیم و دیدیم بخش خوبی از ایده استفاده از الگوریتم Greedy Peeling برای حل مسئله چگال‌ترین زیرگراف نیز از همین مفهوم Core Decomposition آمده.

البته این بهترین تقریب ممکن برای این مسئله نیست و Khuller, Saha در [۱۶] یک الگوریتم $2$ـ تقریب برای این مسئله طراحی کردند که در ابتدا برای تمام $n-k+1$ مقدار ممکن $l \geq k$ برنامه خطی زیر را حل می‌کند که این برنامه خطی با الهام گرفتن از برنامه خطی Charikar در [۷] که مسئله DSG را به طور دقیق مدل می‌کند و ما نیز در بخش۲.۱.۲ به آن اشاره کردیم، نوشته شده.

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} x_e \\
\text{to subject} \quad & x_{u,v} \leq y_u \ \forall uv \in E \\
& x_{u,v} \leq y_v \ \forall uv \in E \\
& \sum_{v \in V} y_v = 1 \\
& y_v \leq \tfrac{1}{l} \ \forall v \in V \\
& x, y \geq 0
\end{aligned}$$

در واقع چون نمی‌دانیم جواب بهینه چه تعداد راس دارد، آنها سعی می‌کنند با اجرا کردن همه این برنامه‌های خطی تعداد راس‌های جواب بهینه را حدس بزنند. جواب بهینه برنامه خطی متناظر با $l$ را $d^*(l)$ بنامید، سپس آنها به ازای هر جواب بهینه $d^*(l)$ یک زیرگراف حداقل $k$ راسی بدست می‌آورند که چگالی آن حداقل $\frac{d^*(l)}{2}$ باشد سپس آنها از میان همه زیرگراف‌های بدست آمده زیرگرافی که بیشترین چگالی را دارد خروجی می‌دهند. حال واضح است اگر زیرگراف بهینه مسئله $l^*$ راسی باشد با قرار دادن $y_i = \frac{1}{l^*}, x_{i,j} = \frac{1}{l^*}$ برای همه راس‌ها و یال‌های آمده در زیرگراف بهینه در برنامه خطی متناظر با $l^*$، اندازه تابع هدف این برنامه خطی برابر با چگالی جواب بهینه می‌شود، پس به یک $2$ـ تقریب می‌رسیم زیرا اگر جواب نهایی الگوریتم را $G_A$ و گراف بدست آمده برای هر $l$ را $G_l$ بنامیم داریم:

$$\forall l \geq k \ : \ d(G_A) \geq d(G_l) \, , \, d(G_{l^*}) \geq \frac{d^*(l^*)}{2} \geq \frac{OPT}{2} \rightarrow d(G_A) \geq \frac{OPT}{2}$$

ممکن است برایتان سوال شود که چرا همین الگوریتم برای مسئله‌ي DKSG کار نمی‌کند. نکته‌ای که وجود دارد این است که زیرگراف $G_l$ای که آنها در این الگوریتم پیدا می‌کنند لزوما $l$ راسی نیست و صرفا بالای $k$ راس دارد برای همین نمی‌شود با حل برنامه خطی بالا برای $l=k$ لزوما به جوابی برای DKSG رسید.

### ۳.۱.۴ مسئله‌ی Densest At Most k Subgraph

مسئله DAMKS به مانند مسائل قبلی در کلاس $\mathcal{NP}$-Hard قرار دارد. با کمی دقت می‌توان دید کاهشی که در قضیه ۱۳ دادیم در اینجا نیز کار می‌کند زیرا در صورتی که گراف یک خوشه با اندازه $k$ داشته باشد همان خوشه چگال‌ترین زیرگراف با

اندازه حداکثر $k$ در این گراف می‌شود و چگالی آن نیز $\frac{k-1}{2}$ خواهد بود، پس همان کاهش که بیان کردیم در اینجا نیز صادق است.

**قضیه ۱۷** *مسئله‌ی $DAMKSG$ یک مسئله‌ی $\mathcal{NP}$-Hard هست و $DAMKSG \leq_p Clique$*

در مورد سختی تقریب این مسئله نیز نتایجی وجود دارد که نشان می‌دهند این مسئله تا حد نزدیکی مانند مسئله DKSG برای تقریب زدن سخت است و اختلاف آنها در حد یک ثابت است. در واقع به طور دقیق‌تر قضیه زیر را از Khuller and Saha [۱۶] داریم که به شرح زیر است.

**قضیه ۱۸** *اگر برای مسئله‌ی $DAMKSG$ یک الگوریتم $\alpha$ـ تقریب داشته باشیم آنگاه یک الگوریتم $4\alpha$ـ تقریب برای مسئله‌ی $DKSG$ خواهیم داشت.*

### ۲.۴ مسائل دارای قید روی همبندی زیرگراف

در دسته از مسائل به دنبال چگال‌ترین زیرگرافی هستیم که در یک سری قیود همبندی هم صدق کند. این نسخه از مسئله چگال‌ترین زیرگراف نسخه مهمی است زیرا یکی از مشکلاتی که مسئله چگال‌ترین زیرگراف دارد این است که زیرگراف پیدا شده در آن با اینکه چگالی بالایی دارد ولی ممکن است دارای یک Single Point of Failure باشد و این یک اتفاق منفی در مخابرات و شبکه‌های کامپیوتری است. بسته به نوع همبندی که ما در نظر داریم می‌توانیم این مسائل را به دو دسته زیر تقسیم کنیم.

۱. مسئله یافتن چگال‌ترین زیرگراف $k$ـ همبند راسی.

۲. مسئله یافتن چگال‌ترین زیرگراف $k$ـ همبند یالی.

این دو مسئله برای اولین بار در [۳] مطرح شدند و در آنجا برایشان یک الگوریتم $4$ـ تقریب ارائه شد. در واقع الگوریتم ارائه شده آنها یک الگوریتم $(\frac{4}{\alpha}, \frac{1}{\alpha})$ـ تقریب بوده که $1 \leq \alpha \leq 2$ یعنی هرچه بخواهیم همبندی بهتر رعایت شده باشد جواب نهایی تقریب بدتری دارد و برعکس.

## ۵ تحلیل

### ۱.۵ راه‌اندازی و پیاده‌سازی

پیاده‌سازی تمام الگوریتم‌ها، تست جنریتورها، نتایج اجرای الگوریتم‌ها به همراه پلات‌ها همگی در گیت‌هاب قرار داده شده‌اند.
تمام الگوریتم‌هایی که در ادامه به آن‌ها اشاره می‌کنیم به زبان پایتون پیاده‌سازی شده و روی یک کامپیوتر مجهز به پردازنده‌ی Intel core i9-11900H @ 2.5GHz و رم 16GB اجرا شده‌اند و زمان‌های بیان شده بر واحد میلی‌ثانیه می‌باشند.
در ادامه، نتایجی را که از اجرای الگوریتم‌های زیر به دست آورده‌ایم را گزارش می‌کنیم:

- الگوریتم Goldberg در بخش ۱
- الگوریتم Charikar در بخش ۲
- الگوریتم Greedy Peeling در بخش ۳
- الگوریتم Iterative Greedy Peeling در بخش ۴

برای پیاده‌سازی max-flow و Charikar که نیاز به حل LP دارد، از لایبرری‌های max-flow و cvxpy استفاده شده‌است.
هم‌چنین در الگوریتم گریدی پیلینگ برای خطی کردن زمان اجرای برنامه از لیست پیوندی دوسویه استفاده شده‌است؛ بدین صورت که به ازای هر درجه، یک لیست پیوندی نگه می‌داریم و در هر مرحله از میزانی که کاندید برای کوچک‌ترین درجه است، شروع به پیمایش کرده تا به درجه‌ای برسیم که لیست آن خالی نباشد و سپس یکی از رئوس آن لیست را حذف کرده، لیست‌ها را تغییر داده و مقدار جدید کاندید کم‌ترین درجه را برابر مقدار درجه‌ی کنونی منهای یک قرار می‌دهیم (در واقع کمینه‌ی درجه پس از حذف این راس حداکثر یک واحد کم‌تر می‌شود) بدین صورت می‌توان الگوریتم را در زمان $\mathcal{O}(n+m)$ پیاده‌سازی کرد.
هم‌چنین در الگوریتم iterative-greedy-peeling از لایبرری heapq برای عملیات‌های اضافه‌کردن و حذف‌کردن کمینه استفاده شده‌است و در نتیجه الگوریتم $\mathcal{O}(T(n+m)log(n))$ خواهد بود.

## ۲.۵ جنریتورها و Testbed

تست‌هایی که برای آنالیز الگوریتم‌ها و نتایج ایجاد شده‌اند از دو روش زیر به دست آمده‌اند:

- بخشی از تست‌ها از دیتاست‌های معروف و real world graphs می‌باشند که ساختار غیر رندوم‌تری دارند و بسیاری ازین تست‌ها، گراف‌هایی با اندازه‌های بزرگ می‌باشند. برای جمع‌آوری این گراف‌ها از منابع متعددی از جمله دیتاست استنفورد استفاده شده‌است. دقت کنید از آن‌جایی که این مسئله اهمیت زیادی در کاربردهای مختلف دارد، مطالعه‌ی و تحلیل روی این تست‌ها حائز اهمیت می‌باشد. همچنین ما درکل سعی کردیم تست‌هایی انتخاب کنیم که در دنیای واقعی این مسئله در تحلیل آنها به کار گرفته می‌شود. برای مثال تعدادی تست مدل کننده گراف جاده‌های بعضی شهرها هستند و یا تعدادی از تست‌ها در واقع گراف شبکه‌های مجازی هستند.
در مجموع 27 تست‌کیس ازین دسته در مجموعه‌ی تست‌ها قرار گرفته‌اند.

- تعدادی از تست‌کیس‌ها به وسیله‌ی جنریتورهای مختلفی که آن‌ها را پیاده کردیم، ایجاده شده‌اند.
جنریتورهای مختلفی که تست‌ها به وسیله‌ی آن‌ها ساخته شده‌اند، به شرح زیر می‌باشد:

۱. 20 تست‌کیس از گراف‌های تنک به صورت تصادفی ایجاد شده‌اند. روش ایجاد این گراف‌ها بدین صورت است که $m$ یال تصادفی بدون تکرار روی گراف $n$ راسی ایجاد شده‌است. برای ایجاد این $m$ یال از یک تناظر یک‌به‌یک از جایگشتی از $\{0, 1, ..., \binom{n}{2}\}$ به تمام یال‌های ممکن استفاده شده‌است.

۲. 20 تست‌کیس از گراف‌های dense به صورت تصادفی ایجاد شده‌اند. روش ایجاد این گراف‌ها بدین صورت است که ابتدا یک جایگشت تصادفی روی تمام یال‌ها داده می‌شود و سپس $m$ تای اول لیست به عنوان یال‌های گراف انتخاب می‌شوند. برای جایگشت تصادفی زدن روی یال‌ها از همان تناظر قسمت قبل استفاده می‌شود.

۳. 10 تست‌کیس از گراف‌هایی که ضریب تقریب برای آن‌ها به 2 میل می‌کند (بدترین ضریب تقریب) با مقادیر $t$ و $p$ مختلف ایجاد کرده‌ایم. ساختار این گراف‌ها را می‌توانید در ۱.۲.۲ مشاهده کنید.

در مجموع، 77 تست‌کیس مختلف برای بررسی الگوریتم‌ها و آنالیز آن‌ها مورد استفاده قرار گرفته‌است.

## ۳.۵ تحلیل

### ۱.۳.۵ نتایج کلی و خروجی‌ها

نتایج به صورت کامل در گیت‌هاب قرار گرفته‌اند و در این‌جا برای تحلیل، نتایج و خروجی‌ها روی مجموعه‌ای منتخب از تست‌ها در جدول ۱.۳.۵ به نمایش داده شده‌اند. برای گرفتن نتایج، الگوریتم goldberg به مدت حداکثر 40 ثانیه اجرا شده‌است و در صورتی که جوابی از الگوریتم دریافت نشد، اجرای الگوریتم متوقف شده‌است و مقدار NaN در خروجی گلدبرگ برای آن تست‌کیس ثبت شده‌است. برای الگوریتم charikar در صورتی که $n$ و $m$ حداکثر 5000 باشند، الگوریتم اجرا شده و در غیر این‌صورت، در خروجی NaN ثبت شده‌است. هم‌چنین الگوریتم greedy++ را روی هر تست با $T = min(\lfloor \frac{10^7}{m} \rfloor, 30)$ اجرا می‌کنیم. برای بررسی ضریب تقریب‌ها، در صورتی که جواب الگوریتم گلدبرگ موجود بود، با آن جواب مقایسه می‌کنیم و در غیر این‌صورت با جواب Iterative Greedy Peeling مقایسه را انجام می‌دهیم و ضریب تقریب Iterative Greedy Peeling را NaN قرار می‌دهیم. در نهایت توجه کنید که ضریب تقریب‌ها به نحوی محاسبه شده‌اند که همواره کوچک‌تر مساوی 1 باشند.
اولین مشاهده‌ای که به صورت طبیعی می‌توان انجام داد این است که مقادیر خروجی الگوریتم‌های goldberg و charikar همواره برابر است و این به این علت است که هر دو الگوریتم، الگوریتم‌های دقیق هستند و جواب آن‌ها، جواب بهینه می‌باشد. هم‌چنین می‌توان مشاهده کرد که ضریب تقریب در هر دو الگوریتم تقریبی مطابق آنچه انتظار داشتیم، همواره بیش‌تر مساوی 0.5 است. البته در عمل می‌توان مشاهده کرد که ضریب تقریب در عمل و خصوصا روی گراف‌های چگال‌تر و تصادفی‌تر بسیار نزدیک به 1 است و این نکته در نتایج قبلی کارهایی که روی این الگوریتم شده‌بود نیز مورد اشاره قرار گرفته بود و ما هم در بخش‌های بعدی بیش‌تر مورد بررسی قرار می‌دهیم.
در رابطه با الگوریتم Iterative Greedy Peeling نیز می‌توان مشاهده کرد که حتی با مقدار $T$ کم (کوچک‌تر مساوی 30) همواره در عمل ضریب تقریب بسیار نزدیک به 1 است. با این‌که در تئوری اثبات شده‌است که الگوریتم به جواب اصلی همگرا می‌شود اما کران بالایی که برای تعداد مراحل در اثبات آورده شده‌است تفاوت چشمگیری با نتایج عملی دارد و این

| testname | \|V\| | \|E\| | goldberg | charikar | greedy peeling | greedy++ | greedy peeling AR | greedy++ AR |
|---|---|---|---|---|---|---|---|---|
| adjnoun | 112 | 425 | 4.7917 | 4.7917 | 4.7778 | 4.7917 | 0.9971 | 1.0 |
| blogcatalog | 10321 | 33983 | 98.2791 | NaN | 98.2791 | 98.2791 | 1.0 | 1.0 |
| com-amazon.ungraph | 334863 | 925872 | 4.8041 | NaN | 3.7576 | 4.5909 | 0.7822 | 0.9556 |
| com-youtube | 1134890 | 2987624 | NaN | NaN | 45.5731 | 45.5865 | 0.9997 | NaN |
| ego-facebook | 4039 | 88234 | 77.3465 | NaN | 77.3465 | 77.3465 | 1.0 | 1.0 |
| ego-twitter | 81306 | 1342296 | NaN | NaN | 68.4142 | 69.6185 | 0.9827 | NaN |
| web-Google | 875713 | 4322051 | NaN | NaN | 27.1792 | 27.7872 | 0.9781 | NaN |
| roadNet-CA | 1965206 | 2766607 | NaN | NaN | 1.7138 | 1.7611 | 0.9732 | NaN |
| roadNet-TX | 1379917 | 1921660 | NaN | NaN | 1.8462 | 1.9322 | 0.9555 | NaN |
| roadNet-PA | 1088092 | 1541898 | NaN | NaN | 1.6878 | 1.7059 | 0.9894 | NaN |
| sparse12 | 54213 | 6312 | 0.9 | NaN | 0.7 | 0.8889 | 0.7778 | 0.9877 |
| sparse13 | 9934 | 4123 | 0.9910 | NaN | 0.9048 | 0.9851 | 0.9130 | 0.9940 |
| sparse2 | 2037 | 432 | 0.9167 | 0.9167 | 0.7 | 0.9167 | 0.7636 | 1.0 |
| dense10 | 981 | 381726 | 389.1193 | NaN | 389.1193 | 389.1193 | 1.0 | 1.0 |
| dense15 | 873 | 153215 | 175.5040 | NaN | 175.5040 | 175.5040 | 1.0 | 1.0 |
| dense13 | 10234 | 298984 | NaN | NaN | 29.2148 | 29.2148 | 1.0 | NaN |
| worst-case2 | 481 | 250 | 0.9523 | 0.9524 | 0.5198 | 0.9524 | 0.5457 | 1.0 |
| worst-case6 | 19279 | 9757 | 0.9958 | NaN | 0.5061 | 0.9958 | 0.5082 | 1.0 |
| worst-case1 | 27 | 18 | 0.9091 | 0.9091 | 0.6667 | 0.9091 | 0.7333 | 1.0 |

Table 1: outputs and overall results(AR stands for approximation ratio.)

سوال به وجود می‌آید که آیا کران داده‌شده برای تعداد مراحل را می‌توان بهبود داد یا خیر. در رابطه با همگرایی این الگوریتم نیز در بخش‌های بعدی به تفصیل پرداخته شده‌است.
در نهایت دقت کنید که در تست‌کیس‌های worst-case حتی با مقادیر $t$ و $p$ کم ضریب تقریب greedy-peeling بسیار نسبت به گراف‌های دیگر تفاوت دارد و با توجه به مقادیر $t$ و $p$ می‌توان به سادگی ضریب تقریب را به 0.5 نزدیک کرد. این مشاهده نتیجه می‌دهد که ضریب تقریبی که برای greedy-peeling ارائه شده‌است، کیپ می‌باشد.

### ۲.۳.۵ زمان اجرا

نتایج به صورت کامل در گیت‌هاب قرار گرفته‌اند و در این‌جا برای تحلیل، نتایج و خروجی‌ها روی مجموعه‌ای منتخب از تست‌ها در جدول ۲.۳.۵ به نمایش داده شده‌اند.
الگوریتم $charikar$ به دلیل حل LP، دارای زمان اجرای بسیار بدی است و بنابراین فقط روی تست‌های کوچک‌تر اجرا شده‌است. در این تست‌ها هم می‌توان بررسی کرد که زمان اجرای charikar تفاوت بسیار زیادی با دیگر الگوریتم‌ها خصوصا الگوریتم‌های تقریبی دارد. برای مثال در تست sparse2 زمان اجرای charikar حدود 4.5 ثانیه است. این در حالی است که زمان اجرای الگوریتم گلدبرگ حدود نیم ثانیه و زمان اجرای گریدی پیلینگ و ایتریتیو گریدی پیلینگ به ترتیب حدود 2 و 100 میلی ثانیه است. این نشان می‌دهد که الگوریتم‌های دیگر بر الگوریتم charikar از نظر زمان اجرا ارجحیت کاملی دارند.
مطابق انتظار الگوریتم greedy-peeling به دلیل خطی‌بودن و استفاده از داده‌ساختارهای سریعی مثل لیست پیوندی دوسویه، زمان اجرای بسیار کم‌تری نسبت به goldberg و Iterative Greedy Peeling دارد و حتی روی گراف‌های بسیار چگال با تعداد رئوس بالا زمان اجرای بسیار بهینه‌ای دارد اما مشکل آن ضریب تقریب آن است که به آن اشاره شد.
در رابطه با الگوریتم‌های goldberg و Iterative Greedy Peeling دقت کنید که زمان اجرای goldberg برابر با $\mathcal{O}(T_{flow}lg(n))$ است که $T_{flow}$ زمان اجرای الگوریتم max-flow در پایتون است که با توجه به [۵] که در آن به الگوریتم مورد استفاده توسط این لایبرری اشاره شده‌است، در بدترین حالت زمان اجرای الگوریتم goldberg می‌تواند $\mathcal{O}(mn^2|C|lg(n))$ باشد که $|C|$ هزینه‌ی کات کمینه است. اما همان‌طور که در [۵] اشاره شده‌است در عمل الگوریتم بسیار سریع‌تر عمل می‌کند. هم‌چنین زمان اجرای الگوریتم Iterative Greedy Peeling، $\mathcal{O}(T(n+m)lg(n))$ است و در نتیجه، غالبا الگوریتم Iterative Greedy Peeling سریع‌تر عمل می‌کند. البته در برخی تست‌ها مثل ego-facebook

| testname | \|V\| | \|E\| | goldberg | charikar | greedy peeling | greedy++ |
|---|---|---|---|---|---|---|
| adjnoun | 112 | 425 | 472.53ms | 440.82ms | 1.01ms | 10.52ms |
| blogcatalog | 10321 | 33983 | 37835.16ms | NaN | 341.53 | 26517.91ms |
| com-amazon.ungraph | 334863 | 925872 | 39977.00ms | NaN | 2988.44ms | 38694.52ms |
| com-youtube | 1134890 | 2987624 | NaN | NaN | 7118.82ms | 42766.46ms |
| ego-facebook | 4039 | 88234 | 2809.32ms | NaN | 92.89ms | 4383.23ms |
| ego-twitter | 81306 | 1342296 | NaN | NaN | 1989.47ms | 30624.26ms |
| web-Google | 875713 | 4322051 | NaN | NaN | 10709.81ms | 49318.26ms |
| roadNet-CA | 1965206 | 2766607 | NaN | NaN | 7118.94ms | 26429.34ms |
| roadNet-TX | 1379917 | 1921660 | NaN | NaN | 5521.85ms | 50446.94ms |
| roadNet-PA | 1088092 | 1541898 | NaN | NaN | 4333.37ms | 31495.55ms |
| sparse12 | 54213 | 6312 | 1594.27ms | NaN | 149.51ms | 3619.99ms |
| sparse13 | 9934 | 4123 | 912.01ms | NaN | 15.64ms | 426.23ms |
| sparse2 | 2037 | 432 | 547.14ms | 4504.56ms | 2.00ms | 100.49ms |
| dense10 | 981 | 381726 | 23708.92ms | NaN | 272.26ms | 26198.75ms |
| dense15 | 873 | 153215 | 6701.25ms | NaN | 104.51ms | 8794.20ms |
| dense13 | 10234 | 298984 | NaN | NaN | 311.09ms | 25965.30ms |
| worst-case2 | 481 | 250 | 502.81ms | 526.64ms | <0.01ms | 15.54ms |
| worst-case6 | 19279 | 9757 | 1227.61ms | NaN | 37.67ms | 1193.81ms |
| worst-case1 | 27 | 18 | 507.59ms | 24.55ms | <0.01ms | 1.01ms |

Table 2: runtime results (values are in terms of miliseconds.)

می‌توان دید که الگوریتم goldberg سریع‌تر است که این به علت تعداد کم رئوس گراف و مقدار زیاد $T$ در الگوریتم Iterative Greedy Peeling است. اما در بسیاری از تست‌ها که اجرای goldberg به علت کند بودن متوقف شده‌است، الگوریتم Iterative Greedy Peeling در زمان حداکثر 50 ثانیه متوقف شده‌است.

### ۳.۳.۵ تحلیل آماری ضریب تقریب greedy-peeling

برای تحلیل آماری ضریب تقریب این الگوریتم، ابتدا تست کیس‌ها را به سه دسته‌ی زیر تقسیم می‌کنیم:

۱. دسته‌ای از تست‌کیس‌ها که مربوط به تست‌کیس‌های worst-case هستند.

۲. دسته‌ای از تست‌کیس‌ها که الگوریتم goldberg برای آن‌ها به جواب نرسیده‌است.

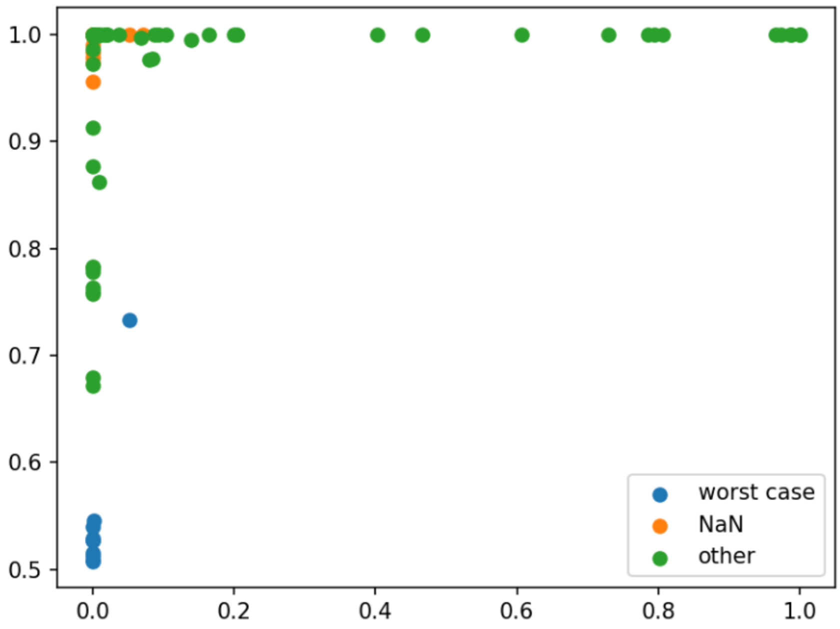
۳. تست‌کیس‌هایی که در دو دسته‌ی فوق قرار نگرفته‌اند.

نتایج به دست آمده در نمودار ۳.۳.۵ به نمایش داده شده‌اند. دقت کنید که بعد $x$ در نمودار، میزان چگال بودن گراف است که با رابطه‌ی $\frac{|E|}{\binom{|V|}{2}}$ به دست آمده است.

همان‌طور که مشاهده می‌شود ضریب تقریب همواره بالاتر مساوی 0.5 است. هم‌چنین در گراف‌های worst-case، غالبا ضریب تقریب بسیار نزدیک به 0.5 است که این نشان می‌دهد که در این دسته از گراف‌ها، ضریب تقریب کیپ می‌شود.

از طرفی دقت کنید که در تست‌کیس‌هایی که الگوریتم goldberg به جواب نرسیده است، ضریب تقریب به شدت نزدیک به 1 می‌شود زیرا از طرفی این گراف‌ها معمولا گراف‌هایی چگال هستند و مطابق با دلایلی که در ادامه بیان می‌شود، ضریب تقریب به 1 نزدیک خواهد شد و هم‌چنین در این گراف‌ها، جواب با Iterative Greedy Peeling مقایسه شده‌است که با توجه به اندازه‌ی بزرگ گراف، مقدار $T$ نسبت به اندازه گراف کوچک می‌شود و این باعث می‌شود در عمل جواب دو الگوریتم نزدیک به هم بشود.

در تست‌کیس‌های دیگر، می‌توان بررسی کرد که با افزایش چگال بودن گراف، ضریب تقریب به 1 نزدیک می‌شود؛ در حالیکه در گراف‌های تنک‌تر ضریب تقریب رفتاری غیر قابل پیش‌بینی‌تر دارد؛ زیرا همانطور که ذکر کرده بودیم الگوریتم greedy-peeling درواقع w_هسته‌های گراف را پیدا می‌کند و این کار در حالتی که در زیرگراف بهینه راس‌های با درجه کم داشته باشیم خوب عمل نمی‌کند؛ به همین دلیل است که رفتار الگوریتم روی گراف‌های تنک‌تر بدتر می‌باشد.



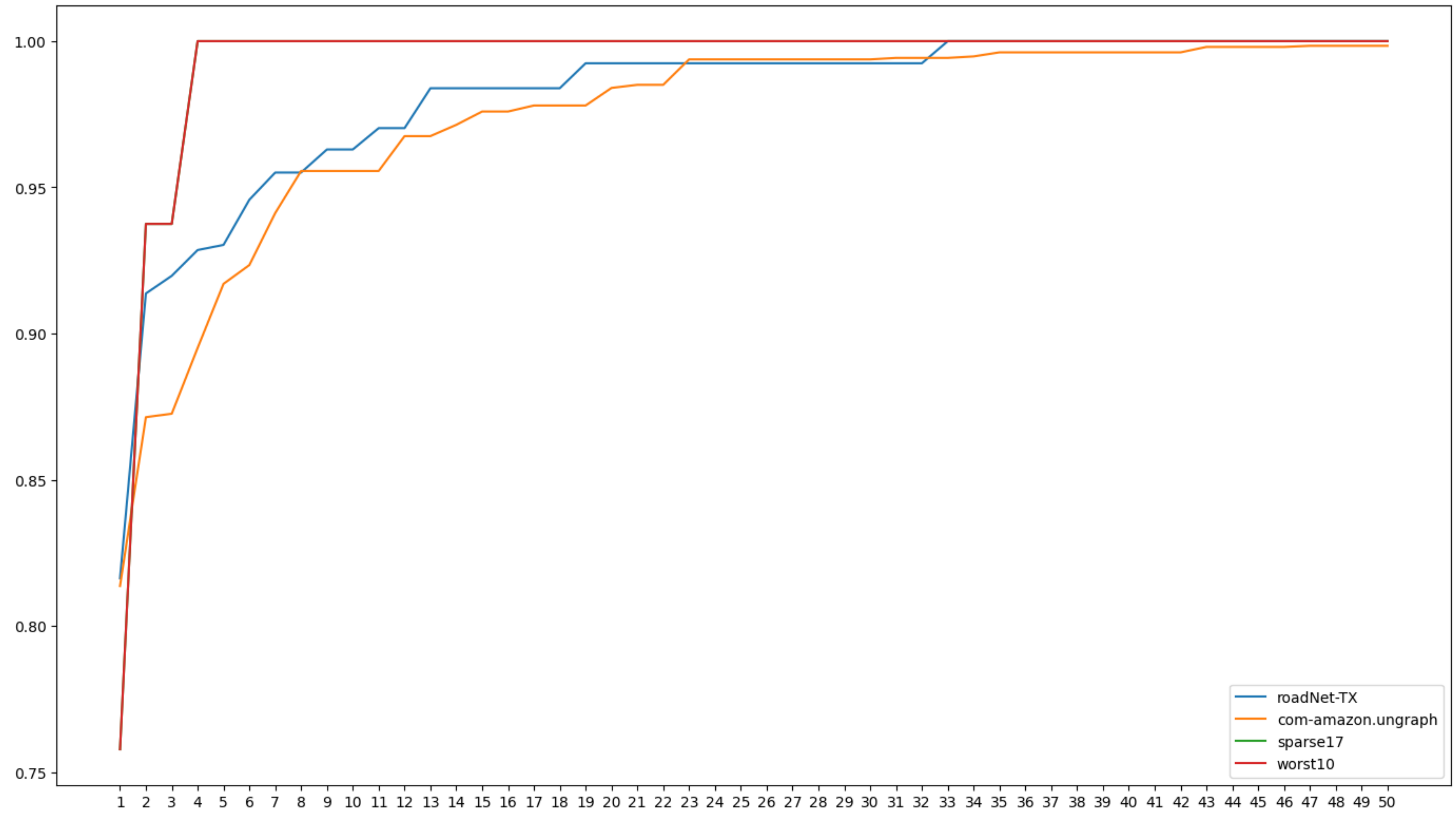

شکل ۳:  ضریب تقریب بر حسب میزان چگال‌بودن گراف

### ۴.۳.۵   همگرایی الگوریتم Iterative Greedy Peeling

همان‌طور که پیش‌تر اشاره شد، الگوریتم Iterative Greedy Peeling با افزایش $T$، به جواب اصلی همگرا می‌شود. در عمل می‌توان مشاهده کرد که الگوریتم با سرعت بیش‌تری نسبت به آن‌چه در تئوری اثبات شده‌است، به جواب بهینه همگرا می‌شود.
برای بررسی این موضوع، ما رفتار الگوریتم را روی تست‌کیس‌های roadnet-TX، com-amazon.ungraph، sparse17 و worst10 به ازای مقادیر مختلف $T$ بررسی کردیم. نتایج این مطالعه در شکل ۴ به نمایش گذاشته شده‌است.
دقت کنید که در تست‌کیس roadnet-TX که الگوریتم goldberg به جواب نرسیده است، جواب را با خروجی الگوریتم iterative greedy peeling روی $T = 50$ مقایسه کرده‌ایم.
همان‌طور که انتظار می‌رود، الگوریتم Iterative Greedy Peeling روی تمام این تست‌کیس‌ها با سرعت بسیار زیادی به جواب همگرا می‌شود و این نشان می‌دهد که با این‌که خود الگوریتم greedy-peeling ممکن است ضریب تقریب نزدیک به 0.5 داشته‌باشد، اما تنها اجرای تعداد محدودی بار نسخه‌ی تعمیم‌یافته‌ی این الگوریتم، می‌توان با سرعت زیادی به جواب بهینه نزدیک شود.

## ۶   نتیجه‌گیری و مسائل باز

همانطور که قبل‌تر هم ذکر کردیم، مسئله چگال‌ترین یک مسئله‌ی مهم در نظریه گراف‌ها و بهینه‌سازی ترکیبیاتی هست که به علت کاربردی که در زمینه‌های مختلف مانند استخراج داده از گراف یا خوشه‌بندی آن و یا در بررسی ساختار DNA و یا شبکه‌های مجازی دارد در سال‌های اخیر مورد توجه زیادی قرار گرفته و الگوریتم‌های متعددی برای آن معرفی شده و نتیجه‌های قبلی آن به طور اساسی بهبود یافته‌است.
نسخه‌های متعدد دیگری از این مسئله وجود دارند مانند چگال‌ترین زیرگراف جهت‌دار، چگال‌ترین زیرگراف در ابرگراف‌ها و چگال‌ترین زیرگراف در فضاهای متریکی که ما در این مقاله نتوانستیم به آنها بپردازیم ولی خواننده می‌تواند برای آشنایی بیشتر با این مسئله‌ها به [۱۸]، [۱۵]، [۱۰] مراجعه کند. همچنین در صورتی که به مطالعه این مسئله در مدل‌های محاسباتی مختلف مثل مدل‌های جویباری، موازی یا مدل‌های توزیع‌شده علاقه‌مند هستید به شما توصیه می‌کنیم [۲]، [۲۱]، [۱۲] و [۹] را مطالعه کنید. ما در این گزارش سعی کردیم تا حد ممکن الگوریتم‌های دقیق و تقریبی مهمی که برای این مسئله وجود دارد را بررسی کنیم و درستی آنها را به زبانی ساده توضیح دهیم. همچنین تلاش کردیم با بررسی الگوریتم‌های مطرح روی




شکل ۴: ضریب تقریب الگوریتم Iterative Greedy Peeling روی تست‌کیس‌های مشخص‌شده و به ازای مقادیر $T$ مختلف

مثال‌های متنوع که در جهان واقعی وجود دارند و هم‌چنین مثال‌های تصادفی، عملکرد این الگوریتم‌ها در حالت‌های مختلف رو تحلیل کنیم و نتایج مختلفی که در عمل از الگوریتم‌ها به دست آمده است را مورد بحث قرار دهیم.

## ۱.۶ مسائل باز

با اینکه گفتیم فعالیت‌های فراوانی در سال‌های اخیر روی این مسئله شکل گرفته ولی هنوز هم سوال‌ها و مشکلات زیادی مرتبط با این مسئله باز باقی مانده‌اند. در این بخش قصد داریم تعدادی از این مسئله‌های باز که به ذهن خودمان رسید یا در منابع معتبر دیدیم را معرفی کنیم.
درست است الگوریتم Iterative Greedy Peeling به جواب بهینه میل می‌کند ولی همچنان بهترین تقریبی که در زمان خطی برای این مسئله داریم ضریب دو دارد، و همانطور که در تحلیل‌های بخش ۵ دیدیم این الگوریتم معمولا خیلی بهتر از دو تقریب عمل می‌کند و فقط در مثال‌های خاصی ضریب تقریب آن نزدیک به دو می‌شود برای همین ما حس می‌کنیم می‌توانیم الگوریتم‌های خطی با ضریب تقریب بهتر از دو داشته باشیم یا حتی با کمی تغییر روی الگوریتم $GreedyPeeling$ کاری کنیم مثال‌های خیلی بد این الگوریتم مانند مثالی که در قضیه۱.۲.۲ دیدیم خنثی شوند و ضریب تقریب این الگوریتم بهتر شود.
همچنین اثبات میل کردن الگوریتم Iterative Greedy Peeling به تازگی و در سال 2022 در [۸] اثبات شد و هنوز جای زیادی برای بررسی و مطالعه بیشتر دارد تا به طور دقیق‌تری به قدرت کامل این روش و نتایج و محدودیت‌های آن پی ببریم.
در نسخه‌های مختلف این مسئله که در بخش۴ نیز سوالات باز زیادی وجود دارد و ضریب تقریب الگوریتم‌هایی که برای این مسائل داده شده‌اند هنوز با کران‌هایی که روی ضرایب تقریب این مسائل داده شده فاصله خیلی زیادی دارد.

# References


[1] Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In *Workshop on Algorithms and Models for the Web-Graph*, 2009.

[2] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proc. VLDB Endow.*, 5(5):454–465, jan 2012.

[3] Francesco Bonchi, David Garc'ia-Soriano, Atsushi Miyauchi, and Charalampos E. Tsourakakis. Finding densest k-connected subgraphs. *Discret. Appl. Math.*, 305:34–47, 2020.

[4] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. Flowless: Extracting densest subgraphs without flow computations. In *Proceedings of The Web Conference 2020*, WWW '20, page 573–583, New York, NY, USA, 2020. Association for Computing Machinery.

[5] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.

[6] Leizhen Cai. Parameterized complexity of cardinality constrained optimization problems. *Comput. J.*, 51:102–121, 2008.

[7] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, 2000.

[8] Chandra Chekuri, Kent Quanrud, and Manuel R Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555. SIAM, 2022.

[9] A. Das Sarma, A. Lall, D. Nanongkai, and A. Trehan. *Dense subgraphs on dynamic networks*, volume 7611 LNCS of *Lecture Notes in Computer Science*, pages 151–165. Springer, January 2012. 26th International Symposium, DISC 2012. ; Conference date: 16-10-2013 Through 18-11-2013.

[10] Hossein Esfandiari and Michael Mitzenmacher. Metric sublinear algorithms via linear sampling. *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 11–22, 2018.

[11] Uriel Feige, Guy Kortsarz, and David Peleg. The dense k -subgraph problem. *Algorithmica*, 29:410–421, 2001.

[12] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrovic. Improved parallel algorithms for density-based network clustering. In *International Conference on Machine Learning*, 2019.

[13] Andrew V. Goldberg. Finding a maximum density subgraph. 1984.

[14] Naga Venkata Chaitanya Gudapati, Enrico Malaguti, and Michele Monaci. In search of dense subgraphs: How good is greedy peeling?, 11 2019.

[15] D.J.-H. Huang and Andrew B. Kahng. When clusters meet partitions: new density-based methods for circuit decomposition. *Proceedings the European Design and Test Conference. ED&TC 1995*, pages 60–64, 1995.

[16] Samir Khuller and Barna Saha. On finding dense subgraphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 597–608, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[17] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. In *J. Algorithms*, 1992.

[18] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. On directed densest subgraph discovery. *ACM Trans. Database Syst.*, 46(4), nov 2021.

[19] Giannis Nikolentzos, Polykarpos Meladianos, Yannis Stavrakas, and Michalis Vazirgiannis. K-clique-graphs for dense subgraph discovery. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 617–633. Springer, 2017.

[20] Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12:141–159, 1982.

[21] Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. *CoRR*, abs/2002.10047, 2020.

[22] Yinyu Ye and Jiawei Zhang. Approximation of dense-n/2-subgraph and the complement of min-bisection. *Journal of Global Optimization*, 25(1):55–73, 2003.