به نام خداوند بخشنده و مهربان

گزارش پروژه پایان ترم طراحی الگوریتم‌ها

استاد درس : جناب آقای دکتر علیرضا باقری

تدریس یار درس : جناب آقای مهندس حمید شهریوری

دانشجویان :

روزبه قاسمی ۹۵۳۱۴۲۴

سپهر عسگریان ابیانه ۹۵۳۱۹۰۱

تابستان ۱۳۹۷

توضیحات :

در ابتدا کمی در مورد الگوریتم های به کار رفته در این پروژه توضیح داده، سپس به شرح کار آن می‌پردازیم.

الگوریتم LPA (Label Propagation Algorithm)

این الگوریتم جز بهترین و سریعترین الگوریتم های تشخیص گراف هاست. همچنین اوردر این الگوریتم نزدیک به زمان خطی است و طراحی نسبتاً ساده ای دارد و قابل فهم است. الگوریتم LPA با تثبیت توالی گره از بروزرسانی برچسب و همچنین تغییر در مکانیزم انتخاب برچسب زمانی که بیش از یک برچسب با حداکثر تعداد گره ها احاطه شده است. در آغاز الگوریتم، یک زیر مجموعه (به طور کلی کوچک) از نقاط داده دارای برچسب (یا طبقه بندی) است .این برچسب ها در طول الگوریتم به نقاط بدون برچسب منتقل می شوند

روش کار این الگوریتم بدین گونه است که هر گره با یک برچسب منحصر به فرد، مانند عدد صحیح و حروف و همچنین در هربار تکرار، هر گره برچسب خود را با برچسب یکی از بزرگترین اعداد گره های همسایه های خود تعویض می‌کند. اگر بیش از یک برچسب با حداکثر تعداد همسایگان خود احاطه شده باشد، به صورت رندوم یکی از آنها را انتخاب می‌کنیم. در اثر تکرار زیاد این فرآیند، گروه های متراکم برچسب های مختلف خود را به برچسب های مشابه تغییر می‌دهند و همچنین گره ها با برچسب های یکسان در یک شبکه مشابه هم گروه می‌شوند.

فرمول به روز رسانی برچسب ها به شرح زیر است:

$$c_i = \arg\max_{l} \sum_{j \in N^l(i)} 1$$

به طوری که $N^l_{(i)}$ با همسایه های $V_{(i)}$ خود برچسب L را جایگذاری می‌کند. در صورت استفاده از گراف وزن دار G وزن یال بین دو گره $V_{(i)}$ و $V_{(j)}$ را با $W_{(ij)}$ نشان می‌دهیم پس فرمول به روز رسانی آن به صورت زیر خواهد بود:

$$c_i = \arg\max_{l} \sum_{j \in N^l(i)} w_{ij}$$

با این حال این الگوریتم نمی‌تواند پس از چند بار تکرار همگرایی آنرا تضمین کند. همچنین در شبکه های بزرگ با تعداد بسیار زیادی از گره ها، ممکن است هر بار به دلیل اینکه شبکه ممکن است تقسیم بندی های مختلفی داشته باشد (به دلیل رندوم بودن عملکرد الگوریتم LPA). در این میان پیداکردن بهینه ترین جواب سخت است بنابراین ثبات این مسئله ضروری است که حل شود. حال برای حل مشکلات پیش آمده از این الگوریتم، دانشمندان یک الگوریتم مشابه این الگوریتم با بهبود مشکلات آن ارائه دادند که در ادامه به آن می‌پردازیم.

الگوریتم NIBLPA(Node Influence Based Label Propagation Algorithm)

این الگوریتم در اصل بهبود یافته الگوریتم قبلی است. تفاوت این الگوریتم با الگوریتم LPA در دو چیز است:

۱- در الگوریتم LPA ، برچسبی که در همسایه ها بیشترین تکرار را داشته باشد انتخاب می شود. اگر چند برچسب با بیشترین تکرار را داشته باشیم به صورت رندوم یکی انتخاب می شود. اما در الگوریتم NIBLPA انتخاب برچسب دقیق تر است و بر اساس $L_{(i)}$ آن ها انتخاب می شود.

۲- در الگوریتم LPA ، در هر مرحله تمامی گره ها به صورت رندوم در هم ریخته می شوند و دوباره برچسب آنها حساب می شود اما در الگوریتم NIBLPA گره ها به ترتیب $N_{(i)}$ که در ادامه گفته می شود ، مرتب می شوند و هر بار تمامی گره ها به ترتیب $L_{(i)}$ بروزرسانی میشود.

این دو مورد وجه تمایز بین دو الگوریتم است که باعث می‌شود الگوریتم NIBLPA نسبت به دیگری عملکرد بهتری داشته باشد.

فرمول محاسباتی این الگوریتم نیز به شرح زیر است:

$$c_i = \arg\max_{l \in l\max} \mathrm{LI}(l)$$

حال به توضیح عملکرد خود پروژه می‌رسیم. در این پروژه از چندین تابع استفاده شده که نیاز به توضیح دارد:

تابع Sort $N_{(i)}$:

تمام $N_{(i)}$ ها را با مرتب سازی حبابی(Bubble Sort)مرتب می‌کند و همچنین محتوای تمام index ها با $N_{(i)}$ مرتب شده است که به ترتیب نزولی است.

تابع NMI :

وظیفه این تابع بدست آوردن NMI باتوجه به اطلاعات داده شده به این تابع است.

تابع K-Shell :

این تابع K-Shell هر گره را محاسبه می کند و می گوید که هر گره در کدام K-Shell قرار دارد.

برای محسبه K-Shell هر گره ،ابتدا گره ها با درجه ۱ حذف می شوند و این فرآیند تا وقتی که هیچ گره دیگری قابل حذف نباشد ادامه می یابد و در گروه K-Shell= 1 قرار می گیرد. به همین ترتیب با درجه ۲ ها و کمتر حذف می شوند و در K-Shell= 2 قرار می گیرند و این تا جایی که تمام شد، فرآیندمان ادامه دارد. گراف TempGraph همان گراف اصلی است و مادامی که زمان change = 1 ادامه می یابد و change=1 اعلام حذف تمام گره ها و پایان را می گوید.

X نشانه درجه ی مورد نظر برای حذف است. هنگامی که یک گره حذف می شود یال های مربوطه نیز حذف می‌شوند پس در گره های مجاور آن ، داده های مربوط به آن گره باید حذف شوند. سپس x یک عدد اضافه می شود که به معنای حذف گره های درجه ۲ و کمتر است. در صورتی که گراف خالی شد ، flag =1 و الگوریتم تمام می شود.

تابع Find $N_{(i)}$:

این تابع NI مربوط به تمامی گره ها را حساب می کند. فرمول محاسبه به صورت زیر است:

$$\mathrm{NI}(i) = Ks(i) + \alpha * \sum_{j \in N(i)} \frac{Ks(j)}{d(j)}$$

آلفا یک عدد بین ۰ تا ۱ است که در فرمول بالا داده شده است که در تابع می توان alpha را قرار داد. Ks(i) همان K-Shell است که در K-Shell Vector محاسبه شده است که جمع $N^{l}_{(i)}$را نگه می دارد و در نهایت $N^{l}_{(i)}$مربوط به هر گره در وکتور $N^{l}_{(i)}$ ریخته می شود.

تابع Algorithm:

این تابع در اصل تابع اصلی و بدنه الگوریتم ماست.. Vector num یک vector است که برای محسبه بزرگترین برچسب های همسایه ی یک گره محاسبه می شود. به عنوان مثال اگر یک lable 2 مشاهده شد ، خانه ی دوم vector یکی اضافه می شود. در آخر خانه های با عدد ماکسیمم در licalc اضافه می شود. Vector ci برچسب هر گره را در خود دارد. در ابتدای کار هر برچسب همان محتوای خود گره است . در صورت تغییر برچسب یک گره ، به خانه ی iام آن ، برچسب جدید اضافه می شود. اگر چند محتوا در licalc حضور داشتند ، با محاسبه LI هر کدام ، lable مربوط به بیشترین LI در ci اضافه می شود. Ci در صورتی به آن lable اضافه می شود که lable جدید با قبلی تفاوت داشته باشد. هنگامی که در یک پیمایش هیچ lable اضافه نشود ، change=1 و از حلقه بیرون می رویم و الگوریتم تمام می شود. در نهایت در finalvec هر خانه lable متناظر خود را دارد.

جدول های بدست آمده

نوع الگوریتم/شماره تست‌کیس	الگوریتم LPA	الگوریتمNIBLPA
Test Case 1	۰.۸۸۷۱۳	۰,۸۷۲۰۴۹۵
Test Case 2	۰.۸۲۱۳۶	۰,۸۱۸۱۷۵۷
Test Case 3	۰.۸۲۲۴۶۸	۰.۸۰۴۲۸۷۳
Test Case 4	۰.۷۷۱۴۸۹	۰.۷۶۴۵۸۵
Test Case 5	۰,۹۹۹۹۸۶	۰,۹۸۶۴۹۱
Test Case 6	۰.۹۸۳۸۳	۰.۹۲۶۷۸
Test Case 7	۰.۶۷۵۸۹	۰.۷۹۷۶۵۱
Test Case 8	۰.۸۷۹۷۱۲	۰.۷۵۹۸۷۶
Test Case 9	۰,۹۹۷۵۹۸۳	۰,۹۹۳۸۲
Test Case 10	۰,۹۹۸۲۳	۰,۹۹۴۲۷
Test Case 11	۰,۹۹۹۴۷	۰,۹۹۸۴۶
Test Case 12	۰,۹۹۹۰۱	۰,۹۹۸۷۷
Test Case 13	۰,۹۹۹۹۴۵	۰,۹۹۸۰۲

براساس NMI

نوع الگوریتم/شماره تست‌کیس	الگوریتم LPA	الگوریتمNIBLPA
Test Case 1	۱۶۵۸	۱۴۲۸
Test Case 2	۱۳۵۷	۱۱۴۳
Test Case 3	۱۶۱۳	۱۴۳۳
Test Case 4	۱۸۰۰	۱۲۷۵
Test Case 5	۵۳۴	۱۴۶۱
Test Case 6	۱۷۵۳	۱۵۳۸
Test Case 7	۲۵۳۵	۱۹۷۸
Test Case 8	۲۹۲۵	۳۱۸۷
Test Case 9	۱۸۷۰۵	۳۱۷۳۶
Test Case 10	۱۶۸۸۷۹	۱۹۵۷۸۰
Test Case 11	۴۶۱۷۸۳۲	۲۵۲۱۳۰۲
Test Case 12	۳۲۳۸۴۷۵	۳۵۲۱۳۶۰
Test Case 13	۲۰۷۳۵۳۲۸	۱۹۸۱۷۲۰۸

بر اساس زمان(میلی ثانیه)

نتیجه گیری

به طور کلی طبق مطالب گفته شده در مقاله، در نتورک های پیچیده و بزرگ هزینه زمانی الگوریتمNIBLPA کمتر از الگوریتم LPA است و بهینه تر است.