دانشکده علوم ریاضی

رشته ریاضی کاربردی، گرایش تحقیق در عملیات

رساله دکتری

بررسی مسأله‌ی $(k,l)-$هسته روی شبکه

نگارنده: سمانه متولی اشکذری

استاد راهنما

دکتر جعفر فتحعلی

استاد مشاور

دکتر سید مهدی زعفرانیه

آذر ۱۳۹۷

در نهایت ادب و احترام

با کمال افتخار

تقدیم به

همه هستی

بهار آفرینش

و رایحه زیبای گل یاس

حضرت مهدی (عج)

پیشکش به:

پدر و مادر مهربانم

ای پدر،

از تو هر چه می گویم باز هم کم می آورم، خورشیدی شدی و از روشنایی ات جان

گرفتم و در ناامیدی ها امیدم بودی و لبریزم کردی از شوق.

اکنون حاصل دستان خسته ات رمز موفقیتم شد.

به خود تبریک می گویم که تو را دارم و دنیا با همه بزرگیش مثل تو را ندارد .....

و تو ای مادر،

ای شوق زیبای نفس کشیدن
ای روح مهربان، هستی ام

تو رنگ شادی هایم شدی و لحظه ها را با تمام وجود برایم زیبا کردی و عمری خستگی ها را به جان خریدی تا اکنون توانستی طعم خوش پیروزی را به من بچشانی.

و همسر محبوبم

که سایه مهربانیش سایه سار زندگیم،

بودنش، همیشه پناهم

و همراهیش آرامشم،

او که اسوه صبر و تحمل بوده و مشکلات مسیر را همواره برایم آسان نمود.

همچنین تقدیم به دو خواهر عزیزم

که وجودشان شادی بخش و نسیم محبتشان همیشه مایه آرامش.

و برادرم

همیشه تکیه گاهم در دریای پر تلاطم زندگی.

و تقدیم به

ستاره های درخشان آسمان زندگی ام

که وجودم بسته به وجودشان

و شادیم در گرو موفقیتشان،

امید و آرزوهایم

ستایش و مهدیار.

سپاس گزاری...

خدای مهربانم را سپاس که زیباترین راه زندگی را به من آموخت. در لحظه لحظه مسیر، همراه و قدم هایم به پشتوانه حضور سبزش محکم و با اطمینان برداشته شد. موهبت های بی نظیرش آسمانم را رنگی و باران نعمتش همواره زمین زندگی ام را سیراب کرد. از او توفیق ماندن زیر چتر علم و دانش و حرکت در راستای خوشنودیش را خواستارم.

بر خود لازم می دانم از تمامی کسانی که مرا در نوشتن این پایان نامه یاری کردند قدردانی و تشکر کنم.

تقدیر و تشکر شایسته از استاد فرهیخته جناب آقای دکتر جعفر فتحعلی که فراتر از راهنمایی این پایان نامه مرا یاری کردند. قطعاً پیشنهادات ارزنده ایشان در موفقیت این تحقیق علمی چشمگیر بوده است.

همچنین تشکر می کنم از استاد مشاور محترمم جناب آقای دکتر مهدی زعفرانیه جهت ارائه نظراتی که در این امر داشتند.

سپاسگزارم از استاد ارجمندم جناب آقای دکتر اردشیر دولتی از دانشگاه شاهد تهران که از مقطع کارشناسی تا داوری این رساله از راهنمایی های ارزشمند ایشان برخوردار بودم.

قدردانی می کنم از جناب آقای دکتر علیرضا ناظمی و دکتر علی عباسی ملایی که زحمت داوری این پایان نامه را بر عهده داشتند.

و با سپاس فراوان از خانواده عزیزم که در تمامی مراحل زندگی همیشه همراه من بودند و فضایی شاد و آرام را جهت تحقیق و پژوهش فراهم آوردند.

از خدای مهربان توفیق روز افزون آنها را مسئلت دارم.

سمانه متولی اشکذری
آذر ۱۳۹۷

تعهدنامه

اینجانب **سمانه متولی اشکذری** دانشجوی دکتری رشته **ریاضی کاربردی علوم ریاضی** دانشگاه شاهرود، نویسنده پایان‌نامه با عنوان **بررسی مسأله‌ی** $(k,l)-$**هسته روی شبکه** ، تحت راهنمایی **جعفر فتحعلی** متعهد می‌شوم:

- تحقیقات در این پایان‌نامه توسط اینجانب انجام شده است و از صحت و اصالت برخوردار است.
- در استفاده از نتایج پژوهش‌های دیگر پژوهش‌گران، به مرجع مورد استفاده استناد شده است.
- مطالب این پایان‌نامه، تا کنون توسط خود، یا فرد دیگری برای دریافت هیچ نوع مدرک یا امتیازی در هیچ‌جا ارایه نشده است.
- حقوق معنوی این اثر، به دانشگاه صنعتی شاهرود تعلق دارد، و مقالات مستخرج با نام '' دانشگاه شاهرود '' یا " Shahrood University of Technology " به چاپ خواهد رسید.
- حقوق معنوی تمام افرادی که در به‌دست آوردن نتایج اصلی پایان‌نامه تاثیرگذار بوده‌اند، در مقالات مستخرج از پایان‌نامه رعایت می‌گردد.
- در تمام مراحل انجام این پایان‌نامه، در مواردی که از موجود زنده (یا بافت‌های آنها) استفاده شده است، ضوابط و اصول اخلاقی رعایت شده است.
- در تمام مراحل انجام این پایان‌نامه، در مواردی که به حوزه اطلاعات شخصی افراد دسترسی یافته (یا استفاده) شده است، اصل رازداری و اصول اخلاق انسانی رعایت شده است.

سمانه متولی اشکذری
آذر ۱۳۹۷

مالکیت نتایج و حق نشر

- تمام حقوق معنوی این اثر و محصولات آن (مقالات مستخرج، کتاب، برنامه‌های رایانه‌ای، نرم‌افزارها و تجهیزات ساخته شده) متعلق به دانشگاه صنعتی شاهرود می‌باشد. این مطلب باید به نحو مقتضی، در تولیدات علمی مربوطه ذکر شود.

چکیده

در این پایان‌نامه مسأله‌ی $(k,l)-$هسته‌ی یک شبکه بیان شده و الگوریتمی برای حل آن روی درخت با وزن مثبت و منفی ارائه می‌شود. این مسأله روی شبکه، یک مسأله‌ی $NP-$سخت است. بنابراین برای پیداکردن $(k,l)-$هسته‌ی یک شبکه، سه الگوریتم ژنتیک طراحی شده است. همچنین نتایج محاسباتی به دست آمده از این الگوریتم‌ها از نظر سرعت و کارایی با هم مقایسه می‌شوند. سپس مسأله‌ی پیدا کردن $(k,l)-$ ۲هسته‌ی یک درخت بررسی شده و الگوریتمی که تعمیمی از الگوریتم $(k,l)-$هسته‌ی یک درخت است، برای حل مسأله‌ی $(k,l)-$ ۲هسته معرفی می‌شود. علاوه بر این، مسأله‌ی پیداکردن هسته‌ی یک شبکه روی گراف‌های بازه‌ای وزن‌دار مطرح شده و روش جدیدی برای حل این مسأله ارائه می‌شود.

کلمات کلیدی:
هسته، مکان‌یابی سرویس‌دهنده، $(k,l)-$هسته، الگوریتم ژنتیک، $(k,l)-$ ۲هسته، درخت‌های بازه‌ای.

لیست مقالات مستخرج از پایان‌نامه

۱. *S.Motevalli Ashkezari and J.Fathali, An efficient algorithm for finding the semi−obnoxious $(k,l)-$ core of a tree, Journal of Mathematical Modeling, University of Guilan, No.* ۲*,* ۲۰۱۶*.*

۲. *S.Motevalli Ashkezari and J.Fathali, On the finding* ۲ *$-(k,l)-$core of a tree with arbitrary real weight, Iranian Journal of Numerical Analysis and Optimization, University of Ferdowsi.*

۳. *S.Motevalli Ashkezari and J.Fathali, Genetic algorithms for finding the semi−obnoxious $(k,l)-$ core of a graph, International Journal of Industrial Engineering : Theory, Applications and Practice [IJIETAP].*

۴. *S.Motevalli Ashkezari and J.Fathali, Finding the $(k,l)-$Core of a tree with pos/neg weight,* ۷*th International Conference of the Iranian Soceity of Operations Research, University of Semnan, May* ۱۴ *−* ۱۵*,* ۲۰۱۴*.*

۵. *S.Motevalli Ashkezari and J.Fathali, FINDING A* ۲ *$-(K,L)-$ CORE OF A TREE,* ۸*th International Conference of the Iranian Soceity of Operations Research, University of Ferdowsi, May* ۲۱ *−* ۲۲*,* ۲۰۱۵*.*

۶. *S.Motevalli Ashkezari and J.Fathali, Solving The Problem Of Finding The $(k,l)-$ core Of A Network With GA,* ۴*th International Conference on Nanotechnology and Basic Science (ICN*۲۰۱۶*), Dubai. Emirates,* ۴ *−* ۵ *February* ۲۰۱۶*.*

۷. *S.Motevalli Ashkezari and J.Fathali, A Genetic Algorithm For Finding The Semi−Obnoxious $(k,l)-$ core Of A Network, The* ۴۶*th Annual Iranian Mathematics Conference, University of Yazd,* ۲۰۱۶*.*

۸. متولی اشکذری.س و فتحعلی.ج، یک الگوریتم خطی برای مساله‌ی پیداکردن هسته درخت‌های بازه‌ای وزن‌دار، مجله‌ی تحقیق در عملیات و کاربردهای آن، دانشگاه لاهیجان، ش. ۲، ۲۰۱۶.

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱

مقدمه

۱.۱ تاریخچه

در جوامع امروزی، بهینه‌سازی در زمینه‌های مختلف اهمیت فراوانی برای صاحبان سرمایه و مدیران کارخانه‌ها دارد. پرکاربردبودن این مسائل توجه بسیاری از محققان را به خود جلب کرده و تحقیقات فراوانی در این زمینه انجام شده‌است. یکی از مسائل مهم بهینه‌سازی، مسأله‌ی مکان‌یابی است. ابتدایی‌ترین نوع این مسأله در قرن هفدهم توسط فرما [۱] مطرح شد. به این صورت که سه نقطه در صفحه داده شده‌است، نقطه‌ـ ی چهارم را به گونه‌ای بیابید که مجموع فاصله آن تا سه نقطه داده‌شده مینیمم شود. توریچلی [۲] در سال ۱۶۴۰ در ایتالیا این مسأله را حل کرد و نشان داد که نقطه بهینه،محل برخورد دوایری است که مثلث‌های متساوی‌الاضلاعی را که بر روی هر ضلع و در بیرون مثلث فعلی قرار دارند در برگرفته‌اند. این نقطه را نقطه توریچلی می‌نامند.

در سال ۱۸۵۷ سیلوستر [۳] مسأله‌ی دیگری را مطرح کرد که در آن هدف پیداکردن نقطه‌ایست که بیشترین فاصله‌اش از سه نقطه داده‌شده کمترین مقدار ممکن باشد. او در سال ۱۸۶۰ با روشی هندسی نشان داد که مرکز دایره‌ای که از این سه نقطه گذشته

---

[۱] *Fermat*

[۲] *Torricelli*

[۳] *Silouster*

است جواب بهینه موردنظر می‌باشد.

اولین تعریف مسأله‌ی مکان‌یابی به صورت کاربردی در سال ۱۹۰۹ توسط آلفرد وبر [۴] آغاز شد. وی مسأله پیداکردن مکان یک سرویس‌دهنده را که مجموع فاصله آن تا چند مشتری کمترین مقدار ممکن است را معرفی کرده و مورد بررسی قرار داد.

بعد از دهه‌ی ۱۹۳۰ با اختراع رایانه و پیشرفت‌هایی که در علوم محاسباتی صورت گرفت روش‌های تکراری برای حل مسائل مکان‌یابی مورد توجه قرار گرفتند. وایزفیلد[۵] یک روش تکراری برای حل مسأله‌ی وبر ارائه داد که هنوز هم مورد استفاده قرار می‌گیرد.

بعد از آن محققین زیادی سعی در بسط و توسعه‌ی مفهوم ارائه شده توسط وبر کردند. به عنوان مثال هاتلینگ[۶][۳۹] مسأله‌ی فروشنده‌ی بستنی در ساحل را مطرح کرد که هدف آن جذب بیشترین تعداد مشتری در یک بازار ساحلی مشترک بین دو یا چند فروشنده است.

مطالعات جدی بر روی مسأله‌ی مکان‌یابی زمانی شروع شد که در سال ۱۹۶۴ حکیمی[۷][۳۲] تابع هدف را به دو صورت کمترین مجموع[۸] و کمترین بیشترین[۹] مطرح کرد. وی مسأله‌ی مکان‌یابی بر روی شبکه را برای پیداکردن مکان گشت‌های پلیس در بزرگراه‌ها و مناطق شهری مورد استفاده قرار داد. او فرض کرد که تعداد سرویس‌دهنده‌ها(گشت‌های پلیس) بیشتر از یکی باشد. بنابراین مشتری‌ها از بین سرویس‌دهنده‌ها، سرویس‌دهنده‌ای را انتخاب می‌کنند که کمترین فاصله را با آن‌ها داشته باشد. به‌این‌ترتیب از اواسط دهه‌ی ۶۰ مکانیابی با پیشرفت شگرفی مواجه شد. با افزایش تنوع مسائل مکان‌یابی، برخی از محققین اقدام به طبقه‌بندی این مسائل کردند. اولین طبقه‌بندی مدل‌های مختلف مکان‌یابی توسط هندلر[۱۰] و میرچندانی[۱۱][۳۷] ارائه شد. پس از آن ایسلت [۱۲] و لاپورت[۱۳][۲۳] و هاماخر[۱۴] و نیکل[۱۵][۳۶] نیز طبقه‌بندی‌های دیگری را انجام دادند. همچنین بررسی‌های دیگری در این زمینه توسط

---

[۴] *Alfred Weber*
[۵] *Weiszfeld*
[۶] *Hotelling*
[۷] *Hakimi*
[۸] *Minisum*
[۹] *Minimax*
[۱۰] *Handler*
[۱۱] *Mirchandani*
[۱۲] *Eiselt*
[۱۳] *Laporte*
[۱۴] *Hamacher*
[۱۵] *Nickel*

کراروپ [۱۶] و پروزن [۱۷] [۴۲]، هانسن [۱۸] و همکاران [۳۸]، میرچندانی و فرانسیس [۴۶] [۱۹]، فرانسیس و همکاران [۲۵]، اون [۲۰] و دسکین [۴۹] [۲۱]، اسکاپارا [۲۲] و اسکاتلا [۵۵] [۲۳] و درنز [۲۴] و هاماخر [۲۱] انجام شده‌است.

لیستی شامل بیش از ۳۴۰۰ مقاله در مورد مسائل مکان‌یابی توسط هال [۲۵] [۳۴] در یک سایت اینترنتی جمع‌آوری شده‌است.

برای پرداختن به انواع مختلف مسائل مکان‌یابی تعاریف زیر را بیان می کنیم.

۲.۱ مفاهیم و تعاریف

تعریف ۱.۲.۱. یک **گراف** [۲۶] شامل یک جفت مجموعه مجزا (V, E) می‌باشد که V مجموعه رئوس و E مجموعه یال‌های آن است.

تعریف ۲.۲.۱. تعداد یال‌های خارج‌شده از هر رأس را **درجه** [۲۷] آن می‌گویند.

تعریف ۳.۲.۱. هر گاه بین دو رأس یک یال وجود داشته باشد این دو رأس را **همسایه** [۲۸] گویند.

تعریف ۴.۲.۱. رأس دلخواه v را در نظر بگیرید. رأس‌های همسایه‌ی آن را فرزندان v بنامید و همین طور رأس‌های همسایه این فرزندان را فرزند فرزند v بنامید. این کار را ادامه دهید تا به آخرین رأس برسید. به رأس v **ریشه** [۲۹] و به رأس قبل از هر **فرزند** [۳۰]، **پدر** [۳۱] آن فرزند گویند.

برای رسیدن به تعریف مسیر ابتدا باید گشت را تعریف کنیم.

---

۱۶ *Krarup*
۱۷ *Pruzan*
۱۸ *Hansen*
۱۹ *Francis*
۲۰ *Owen*
۲۱ *Daskin*
۲۲ *Scaparra*
۲۳ *Scutella*
۲۴ *Drezner*
۲۵ *Hale*
۲۶ *graph*
۲۷ *Degree*
۲۸ *Adjacent*
۲۹ *Root*
۳۰ *Child*
۳۱ *Parent*

تعریف ۵.۲.۱. به دنباله‌ای دلخواه از رئوس که به یکدیگر متصل هستند و رأس تکراری هم می‌تواند داشته باشد **گشت**[۳۲] می‌گویند.

تعریف ۶.۲.۱. به گشتی که هیچ رأس تکراری نداشته باشد **مسیر**[۳۳] می‌گویند.

تعریف ۷.۲.۱. مسیری که با اضافه شدن یک رأس به آن دیگر مسیر نباشد را **مسیر ماکزیمال**[۳۴] گویند.

تعریف ۸.۲.۱. مسیری که ابتدا و انتهای آن بر هم منطبق باشد را **دور**[۳۵] می‌گویند.

تعریف ۹.۲.۱. گراف همبندی که فاقد دور است را **درخت**[۳۶] می‌گویند.

تعریف ۱۰.۲.۱. بیشترین فاصله بین دو رأس از درخت T را **قطر**[۳۷] T نامیده و با d_T نشان می‌دهند.

تعریف ۱۱.۲.۱. هر مسیری که طولش برابر با d_T باشد را **مسیر قطری**[۳۸] می‌گویند.

تعریف ۱۲.۲.۱. یک **شاخه**[۳۹] از رأس v زیردرختی با بیشترین تعداد رأس است که رأس v را به عنوان یک برگ شامل شود.

تعریف ۱۳.۲.۱. فاصله ی دو رأس v و u را با $d(u,v)$ نمایش می‌دهند و برابر با طول کوتاه‌ترین مسیری است که بین این دو رأس وجود دارد.

تعریف ۱۴.۲.۱. فاصله‌ی هر رأس v **از مسیر** p برابر با کوتاه‌ترین مسیری است که بین رأس v و مسیر p وجود دارد و آن را با $d(v,p)$ نمایش می‌دهند، یعنی:

$$d(v,p) = min_{u\in p}\, d(v,u)$$

تعریف ۱۵.۲.۱. فاصله گراف G **از مسیر** p را برابر مجموع فاصله‌ی وزنی رئوس گراف از آن مسیر تعریف می‌کنیم و به صورت زیر نشان می‌دهند.

$$\bar{F}(p) = \sum_{u_i\in V(G)} w_i d(u_i,p)$$

که در آن $w_i \geqslant 0$ وزن رأس v_i است و بیانگر میزان جمعیت یا حساسیت رأس v_i است.

[۳۲] *Walk*
[۳۳] *Path*
[۳۴] *Maximal Path*
[۳۵] *Cycle*
[۳۶] *Tree*
[۳۷] *Diameter*
[۳۸] *Diameter Path*
[۳۹] *Branch*

تعریف ۱.۲.۱۶. اگر مسیر p' بخشی از مسیر p و نه تمام آن را شامل شود یعنی $p' \subset p$ مسیر p' را **زیر مسیر محض** p گویند. واضح است که $\bar{F}(p) < \bar{F}(p')$ است.

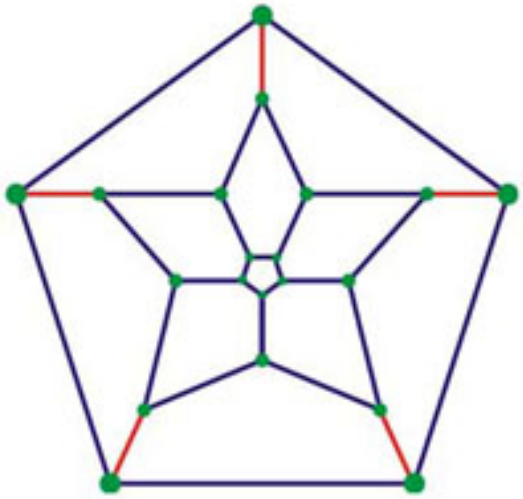
تعریف ۱.۲.۱۷. طوقه یالی است که رأس را به خودش وصل می‌کند.

تعریف ۱.۲.۱۸. گراف ساده گرافی است که بین هر دو رأس از آن حداکثر یک یال وجود داشته باشد و هیچ طوقه‌ای نیز نداشته باشد.

تعریف ۱.۲.۱۹. یک **مسیر همیلتونی**[۴۰] یا مسیر قابل تعقیب، مسیری است که هر راس را دقیقاً یک بار مشاهده کند.

تعریف ۱.۲.۲۰. دور همیلتونی دوری است که هر راس را دقیقاً یک بار مشاهده می‌کند.(به جز راسی که هم به عنوان آغاز و هم پایان می‌باشد. در نتیجه این راس دو بار دیده می‌شود.)

تعریف ۱.۲.۲۱. گراف ساده‌ی G از مرتبه n هرگاه دوری به طول n داشته باشد یا به عبارت دیگر گرافی که دارای دور همیلتونی باشد را **گراف همیلتونی** می‌نامند. یک گراف همیلتونی در شکل ۱.۱ آورده شده است.



شکل ۱.۱: یک گراف همیلتونی

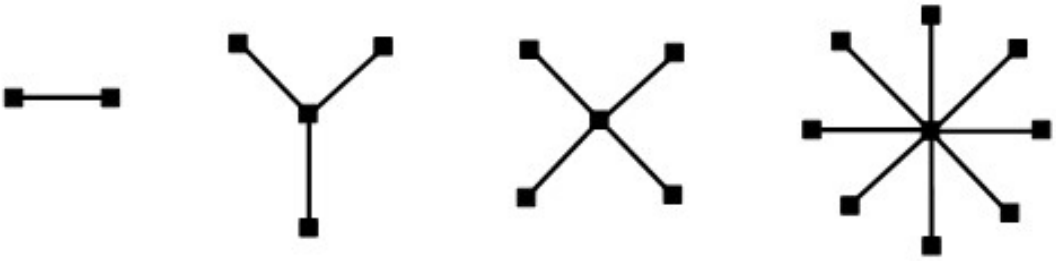
تعریف ۱.۲.۲۲. به هر گراف G با n رأس که در آن یک رأس از درجه‌ی $n-1$ و $n-1$ رأس دیگر از درجه‌ی یک باشند یک **گراف ستاره**[۴۱] می‌گویند.

گراف‌های نشان داده شده در شکل ۲.۱ مثال‌هایی از گراف‌های ستاره هستند. گراف‌های ستاره در حالت کلی درخت می‌باشند.

---

[۴۰] *Hamiltonian Graph*

[۴۱] *Star Graph*



شکل ۲.۱: گراف‌های ستاره

۳.۱ مسائل مکان‌یابی

در حالت کلی مسأله‌ی مکان‌یابی به دو حالت پیوسته و گسسته تقسیم می‌شود.
اگر مکان سرویس‌دهنده‌ها بتواند در هر نقطه‌ای از صفحه قرار بگیرد، مسأله را مسأله‌ی مکان‌یابی پیوسته گویند. برای این حالت راه حل‌هایی برای وقتی که تابع فاصله به صورت یکی از نرم‌های خطی، اقلیدسی و چپیشف باشد در [۲۵] ارائه شده است.
در حالت گسسته یک شبکه مانند $N = (V, E)$ موجود است که تابع فاصله روی آن تعریف می‌شود و مکان سرویس‌دهنده‌ها نیز باید نقطه‌ای از شبکه باشد. ما در این پایان‌نامه حالت گسسته را در نظر می‌گیریم.
مسأله‌ی مکان‌یابی انواع مختلفی دارد که ما در این‌جا، بر اساس نوع سرویس‌دهنده به ذکر چند مورد از آن می‌پردازیم.

- مکان‌یابی نقطه
- مکان‌یابی مسیر
- مکان‌یابی درخت

۱.۳.۱ مکان‌یابی نقطه

ساده‌ترین مسأله‌ی مکان‌یابی همان مسأله‌ای است که فرما مطرح کرد که در ابتدای فصل بیان شد. تعمیمی از آن که به مسأله‌ی فرما ـ وبر معروف می‌باشد به این صورت است که فرض کنید n نقطه‌ی $p_1, p_2, \cdots, p_n$ در صفحه موجودند و به ترتیب دارای وزن‌های $w_1, w_2, \cdots, w_n$ هستند، می‌خواهیم نقطه‌ای مانند x را به گونه‌ای بیابیم که مجموع وزنی فاصله‌ی x تا نقاط موجود کمینه شود. یعنی اگر فاصله x تا p را به صورت $d(x, p)$ نمایش دهیم، آن‌گاه مسأله به صورت زیر خواهد بود:

$min \sum_{i=1}^{n} w_i d(x, p_i)$

مسأله‌ی فوق به مسأله‌ی تک سرویس‌دهنده با کمترین مجموع[۴۲] معروف است. همچنین اگر هدف پیداکردن x به گونه‌ای باشد که فاصله‌ی وزنی x تا دورترین نقطه‌ی موجود کمینه شود، آن‌گاه مسأله را مسأله‌ی تک سرویس‌دهنده‌ای مینیماکس[۴۳] گویند که به صورت زیر است:

$min\ max_{i=1,\cdots,n} w_i d(x, p_i)$

اگر به جای یک نقطه به دنبال چند نقطه باشیم، یعنی چند مکان برای سرویس‌دهنده‌ها به گونه‌ای بیابیم که نزدیک‌ترین نقطه را به نزدیک‌ترین وسیله نسبت داده و بخواهیم مجموع وزنی فاصله‌ها را کمینه کنیم، مسأله را مسأله‌ی چندسرویس‌دهنده با کمترین مجموع[۴۴] و در حالت دیگر آن را مسأله‌ی چندسرویس‌دهنده‌ای مینیماکس[۴۵] گویند. در این حالت اگر $X = \{x_1, x_2, \cdots, x_k\}$ مجموعه سرویس‌دهنده‌ها باشد که باید مکان‌یابی شوند و $d(X, p) = min_{x_i \in X} d(x_i, p)$ آن‌گاه تابع هدف در مسائل کمترین مجموع و مینیماکس به ترتیب می‌تواند به صورت زیر نوشته شود:

$minF(X) = \sum_{i=1}^{n} w_i d(X, p_i)$

و

$minG(X) = max_{i=1,\cdots,n} w_i d(X, p_i)$

واضح است که در حالت چندسرویس‌دهنده علاوه بر پیداکردن مکان سرویس‌دهنده‌ها، مسأله‌ی تخصیص نقاط به سرویس‌دهنده‌ها نیز باید در نظر گرفته شود. در این حالت وقتی تابع هدف به صورت کمترین مجموع یا مینیماکس است، به ترتیب مسأله را مسأله‌ی میانه[۴۶] یا مرکز[۴۷] می‌نامند. همچنین اگر هدف پیداکردن مکان p وسیله‌ی جدید باشد، آن‌گاه این مسائل به ترتیب $p-$میانه یا $p-$مرکز نامیده می‌شوند.
اولین افرادی که به مسأله‌ی مکان‌یابی گسسته با کمترین مجموع توجه کردند کوهن[۴۸] و هامبرگر[۴۹] [۴۳]، حکیمی [۳۲]، مان[۵۰] [۴۴] و بالینسکی[۵۱] [۵] بودند. در میان افراد فوق این حکیمی بود که اصطلاح میانه را بر روی آن نام نهاد.

---

[۴۲] *Single Facility Minisum Problem*
[۴۳] *Single Facility Minimax Problem*
[۴۴] *Multi Facility Minisum Problem*
[۴۵] *Multi Facility Minimax Problem*
[۴۶] *Median*
[۴۷] *Center*
[۴۸] *Kuehn*
[۴۹] *Hamburger*
[۵۰] *Manne*
[۵۱] *Balinski*

مسأله ی $p-$میانه

مسأله‌ی $p-$میانه کاربردهای فراوانی دارد که از آن جمله می‌توان به مکان‌یابی مراکز شبکه‌های ارتباطی کامپیوتری، مراکز توزیع کالا، مراکز اداری، مراکز نظامی، ایستگاه‌های اتوبوس و مراکز پستی اشاره کرد. به وضوح در این نوع مسائل دسترسی آسان به سرویس‌دهنده‌ها از اهداف مهم مکان‌یابی به شمار می‌رود. بنابراین همان‌طور که چرچ و رول[۵۲] در [۱۹] متذکر شده‌اند، مسافتی که مشتری‌ها برای رسیدن به سرویس‌دهنده‌ها طی می‌کنند، می‌تواند معیار مهمی برای سنجش بهینگی باشد. این توجیه در حوزه‌ی عملکرد بسیاری از اماکن عمومی منطقی به نظر می‌رسد. به عنوان مثال مدارس، بیمارستان‌ها، ایستگاه‌های آتشنشانی و بقیه‌ی کاربردهای فوق باید در مکان‌هایی احداث گردند که دسترسی به آن‌ها تا حد امکان تسهیل گردد. به طور مشابه می‌توان این بحث را به مسائل ناخوشایند[۵۳] تعمیم داد. در این مسائل هر چه مسافت سرویس‌دهنده به مشتری‌ها بیشتر باشد مکان سرویس‌دهنده مطلوب‌تر می‌باشد. به عنوان مثال فرودگاه‌ها، مراکز هسته‌ای یا کارخانه‌های زباله‌سوزی که معمولاً در اطراف شهرها تأسیس می‌شوند.

می‌توان به هر یک از مشتری‌ها یک وزن تخصیص داد. به طوری که این وزن بیانگر جمعیت موجود یا حساسیت و اهمیت آن‌ها از لحاظ محیطی و زیست‌شناسی و یا بسیاری از پارامترهای دیگر باشد[۵۴]. به عنوان مثال برای مسائل خوشایند وزن نقاط را مثبت و در مسائل ناخوشایند آن‌ها را منفی در نظر می‌گیریم.

مسأله‌ی $p-$میانه را می‌توان به این شکل تعریف کرد. مکان p سرویس‌دهنده را به نحوی پیدا می‌کنیم که مجموع مسافت وزنی بین مشتری‌ها و این سرویس‌دهنده‌ها کمترین مقدار ممکن باشد. قبل از بیان مدل‌بندی مسأله‌ی $p-$میانه نمادگذاری زیر را داریم:

۱. i اندیس برای مشتری.

۲. j اندیس برای سرویس‌دهنده.

۳. w_i وزن مشتری i ام.

۴. d_{ij} فاصله‌ی بین مشتری iام و سرویس‌دهنده‌ی jام.

۵. p تعداد سرویس‌دهنده‌ها.

۶.

$$x_j = \begin{cases} ۱ & \text{اگر سرویس‌دهنده در مکان } j \text{ قرار بگیرد.} \\ ۰ & \text{در غیر این صورت} \end{cases}$$

[۵۲] *Church and Revelle*

[۵۳] *Undesirable*

۷.

$$y_{ij} = \begin{cases} ۱ & \text{اگر مشتری } i \text{ از سرویس‌دهنده } j \text{ استفاده کند.} \\ ۰ & \text{در غیر این صورت} \end{cases}$$

با این نمادها مسأله‌ی $p-$میانه به صورت زیر مدل می‌شود:

$$F(p) = min \sum_i \sum_j w(i) d_{ij} y_{ij} \quad (۱.۱)$$

$$\sum_j x_j = p \quad (۲.۱)$$

$$\sum_j y_{ij} = ۱ \forall i \quad (۳.۱)$$

$$y_{ij} - x_j \leqslant ۰ \forall i, j \quad (۴.۱)$$

$$y_{ij} \in \{۰, ۱\} \forall j \quad (۵.۱)$$

$$x_j \in \{۰, ۱\} \forall i, j \quad (۶.۱)$$

که رابطه‌ی (۱.۱) بیان می‌کند مجموع وزنی فاصله مشتری‌ها از سرویس‌دهنده‌ها باید تا حد امکان کمینه شود. رابطه‌ی (۲.۱) لزوم واقع‌شدن p سرویس‌دهنده را بیان می‌کند. رابطه‌ی (۳.۱) این اطمینان را می‌دهد که تقاضای تمام رئوس برآورده می‌شود. رابطه‌ی (۴.۱) بیانگر این نکته است که اگر سرویس‌دهنده روی رأس jام واقع نباشد$(x_j = ۰)$ مشتری iام نمی‌تواند از آن سرویس بگیرد$(y_{ij} = ۰)$ و در صورتی که سرویس‌دهنده روی رأس jام باشد$(x_j = ۱)$ مشتری iام می‌تواند از آن سرویس بگیرد یا نگیرد$(y_{ij} = ۰$ یا $y_{ij} = ۱)$.

مسأله‌ی $p-$میانه بر روی یک شبکه یک مسأله‌ی $NP-$کامل[۵۴] است. به هر حال محدود کردن انتخاب p مکان از بین n رأس تعداد حالت‌های ممکن را به

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

محدود می‌کند. اگر n و p خیلی بزرگ نباشند می‌توان به روش‌های شمارشی جواب بهینه را پیدا کرد. برای یک p مشخص، این مسأله یک مسأله‌ی $NP-$سخت[۵۵] است[۲۶]. چنین مرتبه محاسباتی باعث به وجود آمدن الگوریتم‌های فراابتکاری برای حل مسأله‌ی $p-$میانه شده است. با توجه به آن که مدل مسأله‌ی $p-$میانه یک مدل عدد صحیح است روش‌های ابتکاری[۵۶] نیز برای به دست آوردن جواب مورد استفاده قرار گرفته اند(برای مطالعه کامل در این خصوص مراجع [۲۲] و [۵۹] را ببینید).

در سال ۱۹۹۵ گویش[۵۷] و اسریدار[۵۸] مسأله را برای حالت $p = ۲$ روی درخت بررسی

---

[۵۴] $NP - complete$

[۵۵] $NP - hard$

[۵۶] $Heuristic$

[۵۷] $Gavish$

[۵۸] $Sridhar$

کرده و الگوریتمی با پیچیدگی زمانی $O(nlogn)$ ارائه دادند[۲۷]. حالت خاص $p = ۱$ روی درخت در سال ۱۹۷۵ توسط گلدمن[۵۹] بررسی و الگوریتم اکثریت برای آن ارائه شد [۳۰].

۲.۳.۱ مکان‌یابی مسیر

گاهی در مسائل مکان‌یابی، مشتری‌ها در نقاط دوردست هستند و دسترسی آن‌ها به سرویس‌دهنده مشکل است. هدف در این‌گونه مسائل مکان‌یابی دسترسی آسان مشتری‌ها به سرویس‌دهنده است. بنابراین اگر در این مسائل مجموعه مشتری‌ها را رئوس یک گراف و مقدار تقاضای هر مشتری را وزن رأس مربوط به آن در نظر بگیریم، هدف پیداکردن مسیری است که مجموع فاصله‌های وزنی رأس‌های موجود تا آن کمترین مقدار ممکن باشد. چنین مسیری را هسته‌ی درخت می‌گویند. به عبارت دیگر تفاوت این مسائل با مسأله‌ی $p-$میانه این است که در مسأله‌ی $p-$میانه سرویس‌دهنده‌ها بر روی چند نقطه‌ی مجزا و جدا از هم قرار گرفته اند ولی در مسأله‌ی هسته‌ی درخت، سرویس‌دهنده یک وسیله‌ی به هم پیوسته است که بر روی یک مسیر قرار می‌گیرد.
مسأله‌ی پیداکردن هسته روی شبکه و یا درخت به طور جداگانه قابل بررسی و دارای خواص متفاوتی است. حکیمی در سال ۱۹۹۳ نشان داد که پیداکردن مسیر بر روی شبکه یک مسأله‌ی $NP-$سخت است[۳۳]. بنابراین برای حل آن از روش‌های بهینه‌سازی سراسری[۶۰] استفاده می‌شود. مورگان[۶۱] و اسلیتر[۶۲][۴۷] در سال ۱۹۸۰ و بکر[۶۳][۷] در سال ۱۹۹۰ این مسأله را روی درخت در حالتی که وزن همه‌ی رئوس مثبت است، بررسی و الگوریتم‌هایی با زمان خطی برای آن ارائه کرده‌اند.
در مسأله‌ی پیداکردن هسته اگر طول مسیر (هسته) مورد نظر از مقدار مشخصی کمتر شود، هسته را مقید گویند. به عبارت دیگر هدف در این مسأله پیدا کردن مسیر با حداکثر طول l است. محققان زیادی این مسأله را [۸] ، [۴۵] و [۵۱] مورد بررسی قرار دادند.

۳.۳.۱ مکان‌یابی درخت

یکی دیگر از مسائل مکان‌یابی مسأله‌ی پیداکردن یک زیردرخت در شبکه یا درخت اصلی است، به طوری که فاصله‌ی وزنی تمام رئوس تا زیردرخت موردنظر کمترین

[۵۹] *Goldman*
[۶۰] *Global Optimization*
[۶۱] *Morgan*
[۶۲] *Slater*
[۶۳] *Becker*

مقدار ممکن شود. به عبارت دیگر اگر در مسأله‌ی هسته با طول حداکثر l به مسیر بهینه رئوسی را به عنوان برگ اضافه کنیم مسیر تبدیل به زیردرختی از شبکه یا درخت اصلی می‌شود. بنابراین می‌توان علاوه بر محدودیت روی طول مسیر، تعداد برگ‌ها را نیز محدود کرد. پنگ[۶۴] [۵۰] مساله‌ی پیداکردن یک زیر درخت با دقیقاً k برگ را بررسی کرد. به این صورت که هدف در مسأله یافتن زیردرختی با دقیقاً k برگ است که مجموع فاصله ها از همه‌ی رئوس به این زیردرخت کمینه باشد. یک الگوریتم خطی برای حل آن، توسط شیورا[۶۵] و آنو[۶۶] [۵۶] ارائه شده است.

حال اگر هدف یافتن زیردرختی با قطر حداکثر l و تعداد برگ‌های حداکثر k باشد، به طوری که مجموع فاصله‌های وزنی تمام رئوس به این زیردرخت کمترین مقدار ممکن شود، این زیردرخت را $(k,l)-$هسته‌ی درخت می‌نامند. مسأله‌ی پیداکردن هسته‌ی درخت حالت خاصی از آن است که در آن $k=۲$ و هیچ محدودیتی روی طول l نیست. بکر[۸] الگوریتمی کارا برای پیداکردن (k,l)ـ هسته‌ی یک درخت ارائه کرده است. پیچیدگی زمانی این الگوریتم $O(n^۲logn)$ است.

بحث اصلی ما در این پایان‌نامه یافتن $(k,l)-$هسته‌ی یک درخت یا یک شبکه است. در فصل دوم، این مسأله را با وزن مثبت و منفی در نظرمی‌گیریم. روی برخی از ویژگی‌های جواب بهینه‌ی مسأله‌ی (k,l)ـ هسته‌ی یک درخت با وزن مثبت/ منفی بحث می کنیم. همچنین از قضایای ارائه شده در [۶۳] توسط زعفرانیه و فتحعلی استفاده کرده و نشان می‌دهیم زمانی که وزن درخت منفی باشد جواب مسأله‌ی (k,l)ـ هسته‌ی یک درخت با وزن مثبت/ منفی همان جواب مسأله‌ی ۱−میانه است. بر مبنای قضایای ارائه شده الگوریتمی را با پیچیدگی زمانی $O(n^۲logn)$، که در واقع اصلاحی از الگوریتم بکر[۸] است، برای حل مسأله‌ی (k,l)ـ هسته‌ی یک درخت با وزن مثبت و منفی پیشنهاد می کنیم.

در فصل سوم، مسأله‌ی پیداکردن $(k,l)-$هسته‌ی یک شبکه را با وزن مثبت و منفی(نیمه‌ـ ناخوشایند) در نظر می‌گیریم. این مسأله یک مسأله‌ی $NP-$سخت است. بنابراین برای حل آن می‌توان از روش‌های ابتکاری و فراابتکاری استفاده کرد. الگوریتم ژنتیک یک روش جستجوی ابتکاری برای حل مسائل بهینه‌سازی ترکیبیاتی است. این الگوریتم با یک جواب شدنی شروع شده و با بهبوددادن این جواب ادامه می‌یابد. ما برای حل این مسأله سه الگوریتم ژنتیک طراحی و ارائه می‌دهیم. سپس نتایج محاسباتی این سه الگوریتم را بررسی کرده و آن‌ها را از نظر کیفیت جواب و سرعت همگرایی با هم مقایسه می کنیم.

در فصل چهارم ، مسأله را به پیداکردن ۲ $-(k,l)-$هسته‌ی یک درخت در حالت نیمه‌ـ ناخوشایند تعمیم می‌دهیم. یک ۲ $-(k,l)-$هسته‌ی یک درخت، دو زیردرخت منفصل

---

[۶۴] $Peng$
[۶۵] $Shioura$
[۶۶] Uno

T_1 و T_2 از T است که هر یک از آن‌ها باید حداکثر k برگ و قطر حداکثر l داشته باشند، به طوری که مجموع فاصله‌ها از همه‌ی رئوس تا این زیردرخت‌ها کمینه باشد. مدل مسأله را بیان کرده و نشان می‌دهیم، برای درخت در حالت بدون وزن جوابی وجود دارد که هیچ یک از دو زیردرخت $(k,l)-$هسته یک رأس نیستند. سپس در حالتی که مجموع وزن‌های رئوس منفی هستند، نشان می‌دهیم $(k,l)-$هسته‌ی T می‌تواند با حذف‌کردن یالی که متصل به یک برگ است به دست آید. در پایان یک الگوریتم برای این مسأله با وزن مثبت/منفی که تعمیمی از الگوریتم ارائه‌شده توسط بکر در [۸] است، ارائه می‌شود. پیچیدگی این الگوریتم $O(n^3 logn)$ است.

در فصل پنجم، به بررسی مسأله‌ی هسته بر روی گراف‌های بازه‌ای می‌پردازیم. یک گراف بازه‌ای مجموعه‌ای از بازه‌ها است که این بازه‌ها می‌توانند متناظر با رئوس یک گراف باشند به طوری که اگر یالی بین رأس v_i و v_j برقرار باشد، دو بازه متناظر با آن‌ها I_i و I_j دارای اشتراک هستند. ما در این فصل ابتدا هسته را در یک گراف بازه‌ای تعریف کرده، سپس نشان می‌دهیم بازه‌هایی که روی هسته‌ی یک درخت قرار دارند نمی‌توانند بازه‌ای غیر ماکسیمال باشند. همچنین با ارائه خواصی برای این مساله روی درخت‌های بازه‌ای، الگوریتمی خطی با پیچیدگی زمانی $O(n)$ برای پیداکردن هسته‌ی درخت بازه‌ای ارائه می‌کنیم.

فصل ۲

مسأله‌ی $(k,l)-$هسته‌ی یک درخت

۱.۲ مقدمه

مسائل هسته و (k,l)ـهسته در شبکه یک مسأله‌ی $NP-$سخت است. بنابراین این مسائل با الگوریتم‌های ابتکاری و فراابتکاری قابل‌حل هستند و روش دقیقی برای حل این مسائل وجود ندارد. اما الگوریتم‌هایی با زمان چندجمله‌ای برای حل آن‌ها روی یک درخت، توسط محققان ارائه شده است. زعفرانیه و فتحعلی الگوریتم‌های فراابتکاری از جمله مورچگان را برای مسأله‌ی یافتن هسته‌ی یک گراف به کار بردند[۶۲].
همان طور که در فصل قبل توضیح داده شد هسته‌ی یک درخت، یک مسیر از آن درخت است به طوری که مجموع فاصله‌ها از همه‌ی رئوس به آن مسیر کمینه شود. برای حالتی که وزن همه‌ی رئوس مثبت است، مورگان[۱] و اسلیتر[۲][۴۷] در سال ۱۹۸۰ و بکر[۳][۷] در سال ۱۹۹۰ الگوریتم‌هایی با زمان خطی برای پیداکردن هسته‌ی یک درخت ارائه کرده‌اند.
بعد از آن محققان زیادی این مسأله را با قرار دادن یک محدودیت روی طول مسیر در نظر گرفتند [۸] ، [۴۵] و [۵۱]. به عبارت دیگر هدف در این مسأله پیدا کردن مسیر با

---

[۱] $Morgan$
[۲] $Slater$
[۳] $Becker$

حداکثر طول l است. پنگ[۴] [۵۰] مساله را به پیداکردن یک زیر درخت با دقیقاً k برگ توسعه داد. به این صورت که هدف در مسأله یافتن زیردرختی با دقیقاً k برگ است که مجموع فاصله ها از همه‌ی رئوس به این زیردرخت کمینه باشد. یک الگوریتم خطی برای حل آن، توسط شیورا[۵] و آنو[۶] [۵۶] ارائه شده است.
می‌توان هر دو محدودیت (محدودیت روی طول هسته و تعداد برگ‌ها) را با هم در یک مسأله در نظر گرفت. بنابراین هدف در این مسأله یافتن زیردرختی با حداکثر طول l و حداکثر تعداد برگ k است که مجموع فاصله‌ی همه‌ی رئوس تا آن کمترین مقدار ممکن شود. به زیردرخت بهینه در این مسأله (k,l)ـ هسته‌ی درخت می‌گویند. مسأله‌ی پیداکردن هسته‌ی درخت حالت خاصی از آن است که در آن $k = ۲$ و هیچ محدودیتی روی طول l نیست.
بکر[۸] الگوریتمی کارا برای پیداکردن (k,l)ـ هسته‌ی یک درخت ارائه کرده است. پیچیدگی زمانی این الگوریتم $O(n^۲logn)$ است. ایده‌ی او این است که با شروع از درخت T، درختان جدیدی ساخته شود که طول هر مسیر در آن بیشتر از l نباشد. سپس برای هر درخت جدید یک زیربرنامه‌ی حریصانه در نظر می‌گیریم تا زیر درختی شامل ریشه با حداکثر k برگ که مجموع فاصله‌ها را کمینه کند پیدا کنیم.
اگر وزن همه‌ی رئوس در گراف منفی باشد مسأله را مسأله‌ی مکان‌یابی ناخوشایند[۷] می‌گویند. مسأله‌ی میانه از این نوع، مسأله‌ی ماکسیین[۸] است که در آن هدف یافتن نقطه ایست که مجموع فاصله‌ی همه‌ی رئوس تا آن بیشترین مقدار ممکن شود. بنابراین چون رئوس در مسأله‌ی میانه‌ی ناخوشایند دارای وزن منفی هستند بیشترین فاصله همان قدر مطلق کمترین مقدار منفی است. زلینکا[۹][۶۴] نشان داد که جواب مسأله‌ی ۱–ماکسیین در یک برگ از درخت به دست می‌آید. تینک[۱۰][۶۰] نیز یک الگوریتم خطی برای حل این مسأله ارائه کرده است. الگوریتمی نیز برای مسأله‌ی ۱–ماکسیین روی شبکه‌های کلی توسط چرچ[۱۱] و گارفینکل[۱۲] [۲۰] با پیچیدگی زمانی $O(mnlogn)$ آماده شده است، که m تعداد یال‌ها و n تعداد رئوس شبکه می‌باشد. پیچیدگی این الگوریتم توسط تمیر[۱۳] [۵۸] به $O(mn)$ بهبود داده شد.
اگر وزن بعضی از رئوس مثبت و وزن بقیه‌ی رئوس منفی یا صفر باشند مسأله به

---

[۴] *Peng*
[۵] *Shioura*
[۶] *Uno*
[۷] *Obnoxious*
[۸] *Maxian*
[۹] *Zelinka*
[۱۰] *Ting*
[۱۱] *Church*
[۱۲] *Garfinkel*
[۱۳] *Tamir*

عنوان یک مسأله‌ی مکان‌یابی نیمه‌ناخوشایند[۱۴] یا مثبت/ منفی در نظر گرفته می‌شود. بورکارد[۱۵] و کاراراپ[۱۶] [۲۰] نشان دادند که مسأله‌ی ۱−میانه‌ی مثبت/منفی روی یک کاکتوس می‌تواند در زمان خطی حل شود. برای مسأله‌ی $p-$میانه با وزن مثبت/منفی روی شبکه، بورکارد[۱۷] و همکارانش[۱۳] دو مدل مختلف کمینه‌سازی معرفی کردند:

- کمترین مجموع فاصله‌های وزن دار یا MWD[۱۸]
- مجموع کمترین فاصله‌های وزن‌دار یا WMD[۱۹]

آن‌ها با در نظر گرفتن $p = ۲$ الگوریتم‌هایی را با پیچیدگی زمانی $O(n^۲)$ و $O(n^۳)$ به ترتیب برای MWD و WMD ارائه دادند. بنکوزیا[۲۰] و همکاران[۹] پیچیدگی زمانی MWD را به $O(nlogn)$ بهبود دادند. بورکارد و فتحعلی[۱۴] نیز نتایج را برای WMD در حالت $p = ۳$ توسعه دادند. همچنین بورکارد و همکاران[۱۵] یک الگوریتم خطی برای مسأله ی ۱−ماکسیین ارائه کردند.
مسأله‌ی هسته‌ی یک درخت با وزن مثبت/منفی توسط زعفرانیه و فتحعلی[۶۳] مورد بررسی قرار گرفته شده‌است. آن‌ها ثابت کردند زمانی که مجموع وزن‌های رئوس منفی باشد هسته باید یک رأس تنها باشد و نیز زمانی که مجموع وزن‌های رئوس صفر است، هسته‌ای وجود دارد که یک رأس است. آن‌ها همچنین نشان دادند که الگوریتم مورگان و اسلیتر[۴۷] می‌تواند برای پیداکردن هسته‌ی یک درخت با وزن مثبت و منفی نیز به کار برده شود.
در این فصل ما مسأله‌ی پیدا کردن (k,l)ـهسته‌ی یک درخت با وزن مثبت و منفی را در نظر می‌گیریم. روی برخی از ویژگی‌های جواب بهینه‌ی مسأله‌ی (k,l)ـهسته‌ی یک درخت با وزن مثبت/منفی بحث می‌کنیم. همچنین از قضایای ارائه شده در [۶۳] توسط زعفرانیه و فتحعلی استفاده کرده و نشان می‌دهیم زمانی که وزن درخت منفی باشد جواب مسأله‌ی (k,l)ـهسته‌ی یک درخت با وزن مثبت/منفی همان جواب مسأله‌ی ۱−میانه است. بر مبنای قضایای ارائه شده الگوریتمی را که در واقع اصلاحی از الگوریتم بکر[۸] است، برای حل مسأله‌ی (k,l)ـهسته‌ی یک درخت با وزن مثبت و منفی پیشنهاد می‌کنیم. پیچیدگی زمانی این الگوریتم $O(n^۲logn)$ است.

---

[۱۴] $Semio-bnoxious$
[۱۵] $Burkard$
[۱۶] $Krarup$
[۱۷] $Burkard$
[۱۸] $Minimum\ Weighted\ Distances$
[۱۹] $Weighted\ Median\ Distances$
[۲۰] $Benkoczia$

۲.۲ فرمول‌بندی مسأله

فرض کنید $T=(V,E)$ یک درخت باشد که $|V|=n$ و وزن رأس $v_i \in V$ و $w(v_i)$ باشد که می‌تواند مثبت یا منفی باشد (برای سادگی w_i می‌نویسیم) و $a(i,j)$ طول یال (i,j) باشد. بنابراین وزن درخت T، $w(T)=\sum_{i=1}^{n} w_i$ است. همچنین فرض کنید $d(v_i,v_j)$ طول مسیر از v_i به v_j باشد آن‌گاه طول کوتاه‌ترین مسیر بین مسیر p و رأس v، $d(p,v)=min_{u\in p}\, d(u,v)$ است. بیشترین فاصله بین دو رأس از T را قطر T نامیده و با d_T نشان می‌دهند و هر مسیری که طولش برابر با d_T باشد را مسیر قطری می‌گویند.

فرض کنید که $T'=(V',E')$ یک زیردرخت از T باشد. همچنین فرض کنید که $d(v,T')$ کمترین فاصله از $v \notin V'$ تا یک رأس در T'، باشد. مجموع فاصله از T' به همه‌ی رئوس که در V' نیستند را با $d(T')$ نشان می‌دهیم که $DISTSUM$ برای T' می‌نامیم. (k,l)ــ هسته‌ی T زیردرختی مانند T'از T است با حداکثر k برگ و با قطر حداکثر l به گونه‌ای که تابع زیر کمینه شود،

$$F(T')=d(T')=\sum_{v_i\notin V'} w_i d(v_i,T')$$

مسائل نیمه‌ناخوشایند در طراحی شبکه‌های انتقال با سرعت بالا مانند خطوط ریلی، بزرگراه‌ها، قطارهای زیرزمینی، لوله‌های آب و مسیرهای حمل و نقل کاربرد دارند که در آن زیردرخت بهینه می‌تواند ترکیبی از چند مسیر باشد و به خاطر وجود وزن‌ـ های مثبت و منفی، خطوط بین رئوس مطلوب(نواحی شهری) و نامطلوب(انبارهای نظامی) طراحی می‌شوند.

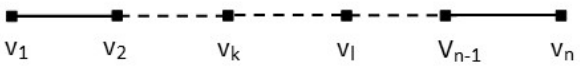
۳.۲ ویژگی‌های مسأله

در این بخش نتایجی را که توسط فتحعلی و زعفرانیه برای هسته‌ی یک درخت در [۶۳] به دست آمده است، بیان کرده و سپس به حالت (k,l)ــ هسته بسط می‌دهیم.
ابتدا فرض کنید وزن درخت T غیر مثبت یا $w(T)\leq 0$ باشد. دو حالت $w(T)<0$ و $w(T)=0$ را به طور جداگانه بررسی می‌کنیم.
زمانی که $w(T)<0$ قضایا و لم‌های زیر را داریم:
قبل از بیان قضایا تعریف زیر را در نظر بگیرید،

تعریف ۱.۳.۲. فرض کنید u یک رأس از درخت T باشد و v رأس همسایه با u باشد. مولفه ی T_{uv} یک زیردرخت به دست آمده از حذف یال uv است به طوری که $u\in T_{uv}$ است.

قضیه ۱.۳.۲. [۶۳] اگر T یک درخت با وزن منفی یعنی $w(T) < ۰$ با شد. آن‌گاه جواب مسأله‌ی ۱–میانه یک هسته از درخت T است و برعکس.

برهان. اثبات را برای حالت ساده‌ای که درخت یک مسیر است، شروع می‌کنیم. فرض کنید $p : v_۱ \_ v_n$ مسیری با وزن $w(p) < ۰$ باشد و فرض کنید $\bar{p} : v_k \_ v_l$ یک زیر مسیر دلخواه p با دو گره متمایز v_k و v_l باشد.



شکل ۱.۲: مسیر $\bar{p}$

به این ترتیب می‌توانیم $F(\bar{p})$ را به صورت زیر بنویسیم،

$$\sum_{i=۱}^{n} w_i d(\bar{p}, v_i) = \sum_{i=۱}^{k-۱} w_i d(\bar{p}, v_i) + \sum_{i=k}^{l} w_i d(\bar{p}, v_i) + \sum_{i=l+۱}^{n} w_i d(\bar{p}, v_i)$$

که $k > ۱$ و $l < n$ و چون $\bar{p}$ شامل رأس‌های v_k تا v_l است بنابراین فاصله‌ی این رأس ها تا $\bar{p}$ صفر است پس می‌توانیم رابطه‌ی فوق را به صورت زیر بنویسیم.

$$\sum_{i=۱}^{n} w_i d(\bar{p}, v_i) = \sum_{i=۱}^{k-۱} w_i d(\bar{p}, v_i) + ۰ + \sum_{i=l+۱}^{n} w_i d(\bar{p}, v_i)$$

اکنون زیر مسیر $\hat{p}$ را که شامل رأس‌های $\bar{p}$ است با وزن‌های جدید به صورت زیر تعریف می‌کنیم:

$$\hat{w}_k = \sum_{i=۱}^{k} w_i,\ \hat{w}_{k+۱} = w_{k+۱},\ \ldots,\hat{w}_{l-۱} = w_{l-۱},\ \hat{w}_l = \sum_{i=l}^{n} w_i$$

در این حالت که وزن‌های رئوس خارج از $\bar{p}$ در برگ‌های $\bar{p}$ جمع شده است، ادعا می‌کنیم که مجموع فاصله‌های وزن‌دار رئوس در $\hat{p}$ به v_k یا به v_l کمتر از صفر است. این ادعا را با برهان خلف اثبات می‌کنیم. فرض کنید مجموع فاصله‌های وزن دار به هر دو رأس v_k و v_l غیر منفی هستند. آن‌گاه نامساوی‌های زیر را داریم:

$$\hat{w}_k d(v_k, v_l) + \hat{w}_{k+۱} d(v_{k+۱}, v_l) + \ldots + \hat{w}_{l-۱} d(v_{l-۱}, v_l) + \hat{w}_l d(v_l, v_l) \geqslant ۰$$

$$\hat{w}_k d(v_k, v_k) + \hat{w}_{k+۱} d(v_{k+۱}, v_k) + \ldots + \hat{w}_{l-۱} d(v_{l-۱}, v_k) + \hat{w}_l d(v_l, v_k) \geqslant ۰$$

با جمع‌کردن نامساوی‌های فوق نامساوی زیر به دست می‌آید:

$$(\hat{w}_k + \hat{w}_{k+۱} + \ldots + \hat{w}_{l-۱} + \hat{w}_l) d(v_k, v_l) \geqslant ۰$$

چون $d(v_k, v_l) > ۰$ و $(\hat{w}_k + \hat{w}_{k+۱} + \ldots + \hat{w}_{l-۱} + \hat{w}_l) = \sum_{i=l}^{n} w_i$ داریم:

$\sum_{i=l}^{n} w_i \geqslant ۰$

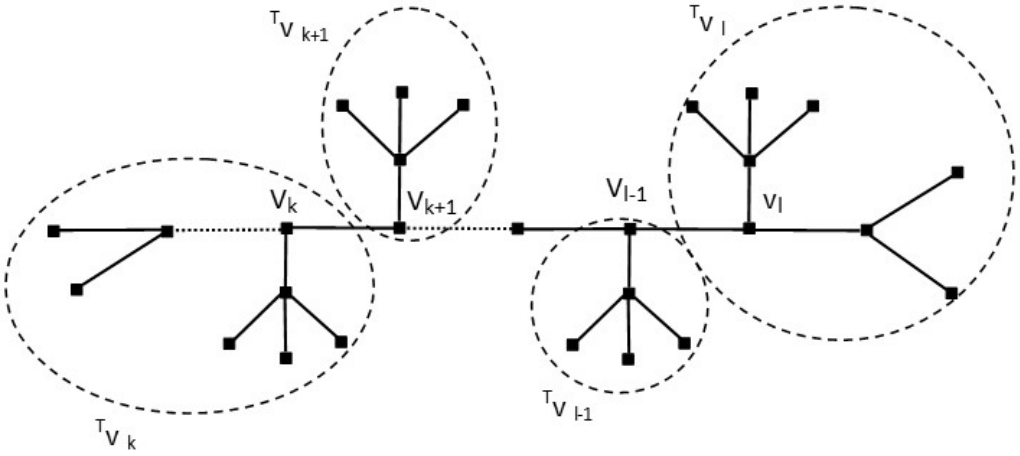
که با فرض $\sum_{i=l}^{n} w_i < ۰$ در تناقض است.
بنابراین مجموع فاصله‌ها تا رئوس v_k یا v_l کمتر از صفر است. بدون از دست‌دادن کلیت، فرض می‌کنیم این حالت در v_k اتفاق بیفتد. بنابراین

$F(v_k) = F(\bar{p}) + \sum_{i=k}^{l} \hat{w}_i d(v_i, v_k) < F(\bar{p})$

چون $\bar{p} : v_k \_ v_l$ یک زیرمسیر دلخواه مسیر p است، هر هسته‌ی p باید یک رأس باشد(زیرا همان‌طور که مشاهده شد $F(v_k)$ از $F(\bar{p})$ کمتر بود).
حال این نتیجه را به درخت دلخواه T با $w(T) < ۰$ تعمیم می‌دهیم.
فرض کنید $\bar{p} : v_k \_ v_l$ یک مسیر دلخواه درخت T باشد که دو رأس v_k و v_l متمایز هستند. مسیر $\hat{p}$ را با رأس‌های مشابه $\bar{p}$ و وزن‌های جدید زیر می‌سازیم.

$\hat{w}_k = \sum_{i \in T_{v_k}} w_i,\ \hat{w}_{k+۱} = \sum_{i \in T_{v_{k+۱}}} w_i,\ \ldots,\hat{w}_{l-۱} = \sum_{i \in T_{v_{l-۱}}} w_i,\ \hat{w}_l = \sum_{i \in T_{v_l}} w_i$

که T_{v_i} و $i = k, \ldots, l$ زیردرخت‌های T هستند که با حذف یال‌های $\bar{p}$ از درخت T به دست می‌آیند، به طوری که $v_i \in T_{v_i}$.



شکل ۲.۲: مسیر $\bar{p}$

مانند حالت قبل داریم:

$F(v_k) = F(\bar{p}) + \sum_{i=k}^{l} w(T_{uv_i}) d(v_i, v_l) < F(\bar{p})$

یا

$F(v_l) = F(\bar{p}) + \sum_{i=k}^{l} w(T_{uv_i}) d(v_i, v_l) < F(\bar{p})$

بنابراین $F(v_k) < F(\bar{p})$ یا $F(v_l) < F(\bar{p})$.
چون $\bar{p} : v_k \_ v_l$ یک رأس دلخواه از درخت T است، هسته باید یک رأس باشد و از میان همه‌ی رأس‌ها بهترین تابع هدف متعلق به ۱–میانه است. بنابراین مسأله‌ی پیداکردن هسته به مسأله‌ی پیداکردن ۱–میانه کاهش پیداکرده است. به طور معکوس مسأله‌ی پیداکردن ۱–میانه معادل با پیداکردن هسته است. □

نتیجه ۱.۳.۲. اگر وزن درخت T صفر باشد، آن‌گاه هسته‌ای وجود دارد که جواب مسأله‌ی ۱–میانه است.

لم ۱.۳.۲. [۶۳] اگر T یک درخت با وزن غیرمثبت باشد، دو حالت زیر ممکن است اتفاق بیفتد:

- هسته یک برگ است.
- هسته یک رأس داخلی u است که $w(u) \geqslant ۰$ و $w(T_{uv}) \leqslant ۰$ برای هر رأس v که با u همسایه است.

برهان. طبق قضیه ۱.۳.۲ و نتیجه ۱.۳.۲ هسته‌ی درخت با وزن $w(T) \leqslant ۰$ یک رأس یا همان ۱–میانه است. اگر هسته روی یک برگ قرار گیرد که لم برقرار است. درغیراینصورت فرض کنید که یک رأس داخلی باشد که هسته(۱–میانه) است. برای هر رأس v_{ik} که همسایه با u است و $k = ۱, \ldots, m$ داریم

$$F(v_{ik}) - F(u) = [w(u) + \sum_{j=۱, j\neq k}^{m} w(T_{uv_{i_j}}) - w(T_{uv_{i_k}})]d(u, v_{ik}) \geqslant ۰$$

بنابراین،

$$w(u) + \sum_{j=۱, j\neq k}^{m} w(T_{uv_{i_j}}) - w(T_{uv_{i_k}}) \geqslant ۰ \qquad (۱.۲)$$

چون

$$w(T) = w(u) + \sum_{j=۱}^{m} w(T_{uv_{i_j}}) \leqslant ۰$$

داریم،

$$w(T_{uv_{i_j}}) \leqslant ۰$$

و چون رأس v_{ik} یک رأس دلخواه همسایه با u است، فرض می‌کنیم،

$$w(T_{uv_{i_k}}) = max\{w(T_{uv_{i_j}}) \mid v_j \text{ با } u \text{ همسایه است}\}$$

بنابراین

$$\sum_{j=۱, j\neq k}^{m} w(T_{uv_{i_j}}) - w(T_{uv_{i_k}}) \leqslant ۰$$

از نامساوی فوق و نامساوی ۱.۲ داریم،

$$w(u) \geqslant ۰$$

□

نکته: اگر وزن‌های رئوس درخت T منفی باشند، لم ۱.۳.۲ بیان می‌کند که جواب مسأله‌ی ۱–میانه یک برگ است. این نتیجه توسط زلینکا [۶۴] نشان داده شده است.

لم ۲.۳.۲. [۶۳]اگر T یک درخت با وزن منفی یعنی $w(T) \leqslant ۰$ با شد، u یک برگ و $w(T) - ۲w(u) > ۰$، آن‌گاه u یک هسته یا ۱–میانه نیست.

برهان. فرض کنید v رأس همسایه با u باشد، آن‌گاه

$$F(v) = F(u) + (۲w(u) - w(T))d(u,v)$$

بنابراین داریم،

$$F(v) < F(u)$$

یعنی u نمی‌تواند هسته‌ی درخت T باشد. □

حال اگر مجموع وزن رئوس درخت مثبت باشد لم زیر را داریم.

لم ۳.۳.۲. [۶۳] فرض کنید T یک درخت با $w(T) > ۰$ با شد و u ۱–میانه‌ی درخت باشد. اگر رأس v همسایه با u باشد، به طوری که $w(T_{uv}) > ۰$ آن‌گاه هر هسته‌ای بیشتر از یک رأس دارد.

برهان. فرض کنید p مسیری از u به v باشد، آن‌گاه

$$F(p) = F(u) - w(T_{uv})d(u,v)$$

با توجه به نامساوی $w(T_{uv}) > ۰$ داریم:

$$F(p) < F(u)$$

و چون u، ۱–میانه است، پس

$$\forall \ z \in v \qquad F(p) < F(z)$$

بنابراین هر هسته‌ای بیشتر از یک رأس دارد. □

نکته: شرط $w(T_{uv}) > ۰$ در لم فوق شرط کافی است و لازم نیست. یعنی ممکن است $w(T_{uv}) < ۰$ در حالی که هر هسته بیشتر از یک رأس داشته باشد.
نکته: اگر درخت T وزن مثبت داشته باشد یعنی $w(T) > ۰$، همیشه این طور نیست که هسته بیشتر از یک رأس داشته باشد.
ما قضایا و لم‌های فوق را که مربوط به هسته‌ی یک درخت است، برای به دست آوردن

نتایجی که منجر به یافتن الگوریتمی برای پیداکردن (k,l)ـ هسته‌ی یک درخت است به کار می‌بریم.

قضیه ۲.۳.۲. اگر T یک درخت با وزن منفی یعنی $w(T) < ۰$ با شد، آن‌گاه جواب مسأله‌ی ۱−میانه یک (k,l)ـ هسته از درخت T است و برعکس.

برهان. از آن جایی که در مسأله‌ی (k,l)ـ هسته هدف ما پیداکردن یک هسته با حداکثر k برگ و با قطر حداکثر l است اگر ما یک ۱−میانه را به عنوان هسته در نظر بگیریم، آن‌گاه با اضافه‌کردن هر رأس دیگری به هسته مقدار تابع هدف افزایش خواهد یافت. بنابراین قضیه برقرار است. □

برای حالت $w(T) = ۰$ نیز در فوق و همچنین در [۶۳] نشان داده شده است، هسته‌ای وجود دارد که ۱−میانه هم است. اما در این حالت ممکن نیست که یک (k,l)ـ هسته‌ای که ۱−میانه نیز باشد وجود داشته باشد. در واقع ممکن است بیشتر از یک رأس داشته باشد. مثال ۳.۵.۲ این حالت را نشان می‌دهد.
فرض کنید u یک رأس از درخت T و v رأس همسایه با آن باشد. آن‌گاه فرض کنید T_{uv} زیردرخت به دست آمده از حذف کردن یال $[u\ v]$ باشد به طوری که $v \in T_{uv}$. در حالتی که $w(T) < ۰$، چون هم هسته و هم (k,l)ـ هسته هر دو ۱−میانه هستند. بنابراین مشابه با ویژگی‌هایی که در [۶۳] برای حالت هسته اثبات شده است، می‌توان لم زیر را نتیجه گرفت.

لم ۴.۳.۲. فرض کنید T یک درخت با وزن منفی باشد. دو حالت ممکن است اتفاق بیفتد:

۱. (k,l)ـ هسته(۱−میانه) یک برگ است.

۲. (k,l)ـ هسته (۱−میانه) یک رأس داخلی u است به طوری که برای هر رأس v همسایه با u، $w(u) \geq ۰$ و $w(T_{uv}) \leq ۰$.

لم ۵.۳.۲. فرض کنید T یک درخت با $w(T) \leq ۰$ باشد. اگر u یک برگ باشد که $w(T) - ۲w(u) > ۰$ آن‌گاه u یک (k,l)ـ هسته نیست.

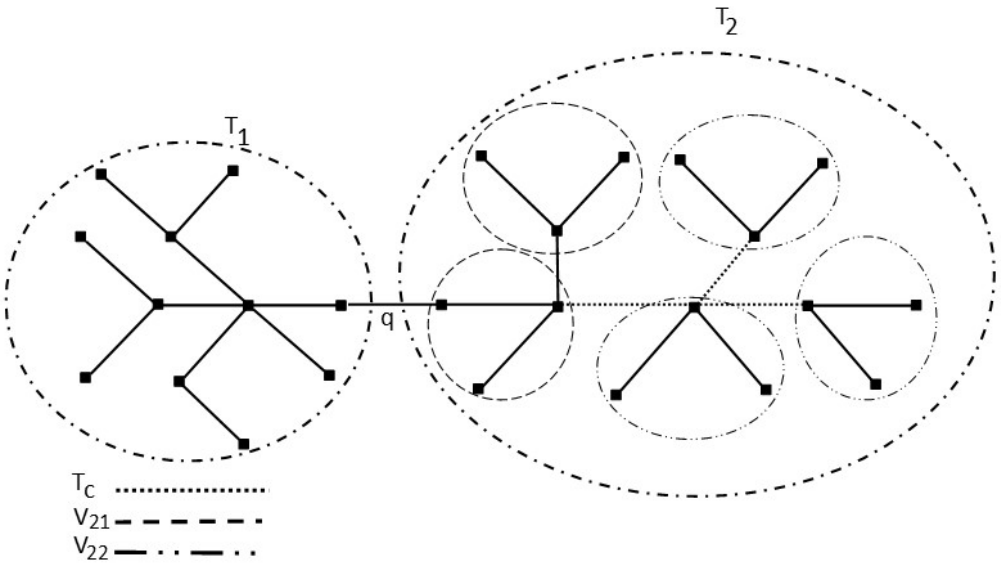
برهان. فرض کنید که v یک رأس همسایه با u باشد، آن‌گاه $d(v) = d(u) + (۲w(u) - w(T))d(u,v)$. بنابراین $d(v) < d(u)$ و درنتیجه u نمی‌تواند یک (k,l)ـ هسته‌ی درخت T باشد. □

اکنون درخت با وزن مثبت را در نظر بگیرید. دوباره از آن جایی که هسته حالت خاصی از (k,l)ـ هسته است، مشابه با ویژگی هسته در [۶۳]، لم زیر را می‌توانیم بیان کنیم.

لم ۶.۳.۲. فرض کنید T یک درخت با $w(T) > ۰$ و u یک ۱–میانه باشد. اگر یک رأس v همسایه با u وجود داشته باشد به طوری که $w(T_{uv}) > ۰$ و $d(u,v) \leq l$، آن‌گاه هر (k,l)ـ هسته بیشتر از یک رأس دارد.

برهان. در اثبات لم ۳.۳.۲ نشان داده شد که تابع هدف هسته برای یال $[u,v]$ کمتر از رأس تنهای u است. بنابراین چون $d(u,v) \leq l$ با اضافه کردن v به u ۱–میانه مقدار تابع هدف برای (k,l)ـ هسته کاهش می‌یابد. بنابراین (k,l)ـ هسته باید بیشتر از یک رأس داشته باشد. □

قبل از بیان قضیه‌ی بعد تعاریف زیر را داریم.
فرض کنید q یالی از درخت T باشد و $T_۱$ و $T_۲$ دو زیردرخت از T باشند که از حذف یال q به دست آمده‌اند. همچنین فرض کنید T_c یک زیردرخت در $T_۲$ و x نزدیک‌ترین رأس $T_۱$ به T_c باشد. ما رئوس $T_۲$ را به سه قسمت $V_{۲۱}$، $V_{۲۲}$ و V_c تقسیم می‌کنیم، به طوری که $V_{۲۱}$ مجموعه رئوس $T_۲$ است که کوتاه‌ترین مسیر از آن‌ها به x از T_c نمی‌گذرد و $V_{۲۲}$ مجموعه رئوس $T_۲$ است که کوتاه‌ترین مسیر از آن‌ها به x از T_c می‌گذرد. همچنین فرض کنید V_c مجموعه رئوس T_c باشد(شکل ۳.۲ را ببینید) و فرض کنید s نزدیک‌ترین رأس از T_c به x باشد. ما طول مسیر از رأس v_i در T_c به s را با l_{vi}نشان می‌دهیم و $u(v_i)$ نزدیک‌ترین رأس در T_c به v_i است.



شکل ۳.۲: تقسیم بندی درخت T

قضیه ۳.۳.۲. فرض کنید $T = (V,E)$ یک درخت با $W(T) > ۰$ و q یک یال از T باشد. همچنین فرض کنید با حذف‌کردن q دو زیردرخت $T_۱$ و $T_۲$ به دست آیند به طوری که $W(T_۱) > W(T_۲)$. اگر T_c یک زیردرخت از $T_۲$ و x نزدیک‌ترین رأس از $T_۱$ به T_c باشد به

طوری که

$$d(x,T_c)\geq\frac{\sum_{v_i\in T_c}w(v_i)l_{vi}+\sum_{v_i\in V_{21}}w(v_i)l_{u(v_i)}}{W(V_1)-W(V_{22})-W(c)}$$

و $W(V_{21})>0$ آن گاه T_c یک (k,l)ـ هسته‌ی درخت T نیست.

برهان. فرض کنید T_c، (k,l)ـ هسته و x نزدیک‌ترین رأس از T_1 به T_c باشد. آن‌گاه:

$$F(T_c)=\sum_{v_i\in V}w(v_i)d(v_i,T_c)=\sum_{v_i\in T_1}w(v_i)d(v_i,T_c)+\sum_{v_i\in T_2}w(v_i)d(v_i,T_c)$$

$$=\sum_{v_i\in T_1}w(v_i)[d(v_i,x)+d(x,T_c)]+\sum_{v_i\in T_2}w(v_i)d(v_i,T_c)=W(V_1)d(x,T_c)$$

$$+\sum_{v_i\in T_1}w(v_i)d(v_i,x)+\sum_{v_i\in V_{21}}w(v_i)d(v_i,T_c)+\sum_{v_i\in V_{22}}w(v_i)d(v_i,T_c)$$

و

$$F(x)=\sum_{v_i\in V}w(v_i)d(v_i,x)=\sum_{v_i\in T_1}w(v_i)d(v_i,x)+\sum_{v_i\in T_2}w(v_i)d(v_i,x)$$

$$=\sum_{v_i\in T_1}w(v_i)d(v_i,x)+\sum_{v_i\in V_{21}}w(v_i)d(v_i,x)+\sum_{v_i\in T_c}w(v_i)[l_{vi}+d(T_c,x)]+$$

$$\sum_{v_i\in V_{22}}w(v_i)[d(v_i,T_c)+l_{u(v_i)}+d(T_c,x)]=\sum_{v_i\in T_1}w(v_i)d(v_i,x)+\sum_{v_i\in V_{21}}w(v_i)d(v_i,x)$$

$$+[W(c)+W(V_{22})]d(T_c,x)+\sum_{v_i\in T_c}w(v_i)l_{vi}+\sum_{v_i\in V_{22}}w(v_i)l_{u(v_i)}$$

از آن جایی که

$$d(x,T_c)\geq\frac{\sum_{v_i\in T_c}w(v_i)l_i+\sum_{v_i\in V_{21}}w(v_i)l_{u(v_i)}}{W(V_1)-W(V_{22})-W(c)}$$

بنابراین

$$[W(V_1)-W(V_{22})-W(c)]d(x,T_c)\geq\sum_{v_i\in T_c}w(v_i)l_i+\sum_{v_i\in V_{21}}w(v_i)l_{u(v_i)}$$

و

$$W(V_1)d(x,T_c)\geq\sum_{v_i\in T_c}w(v_i)l_{vi}+\sum_{v_i\in V_{21}}w(v_i)l_{u(v_i)}+[W(V_{22})+W(c)]d(x,T_c).$$

بنابراین $F(T_c)\geq F(x)$ که با تعریف زیردرخت T_c به عنوان (k,l)ـ هسته در تناقض است. □

از آن جایی که یک هسته حالت خاصی از یک (k,l)ـ هسته است، بنابراین قضیه‌ی ۳.۳.۲ برای هسته نیز صادق است.

قضیه ۴.۳.۲. فرض کنید $T=(V,E)$ یک درخت با $w(T)\geq 0$ باشد و T_1 زیردرختی از T باشد که $d_{T_1}<l$ و تعداد برگ‌هایش کمتر از k باشند. اگر $w(T\setminus T_1)>0$ آن گاه T_1 یک (k,l)−هسته‌ی T نیست.

برهان. فرض کنید $T_1^c=T\setminus T_1$. از آن جا که $w(T_1^c)>0$ است، با حذف T_1 از T مولفه‌ای مانند T_2 وجود دارد که $w(T_2)>0$. اگر $u\in T_2$ رأس همسایه با T_1 باشد و $T_3=T_1\cup\{u\}$. آن گاه

$$F(T_1)=\sum_{v_i\in T_1^c\setminus T_2} w_i d(v_i,T_1)+\sum_{v_i\in T_2} w_i d(v_i,T_1)$$

$$=\sum_{v_i\in T_1^c\setminus T_2} w_i d(v_i,T_1)+\sum_{v_i\in T_2} w_i (d(v_i,u)+d(u,T_1))$$

$$=F(T_3)+d(u,T_1)w(T_2)>F(T_3).$$

بنابراین T_1 یک (k,l)−هسته‌ی T نیست. □

۴.۲ الگوریتم

الگوریتمی توسط بکر با پیچیدگی زمانی $O(n^2 logn)$ برای پیداکردن یک (k,l)ـ هسته‌ی یک درخت با وزن مثبت در [۸] ارائه شده است. ما الگوریتم آن‌ها را تغییر می‌دهیم تا برای درختان با وزن مثبت و منفی قابل کاربرد باشد.

فرض کنید P_{uv} مسیر بین رئوس u و v در درخت T باشد و T^u درخت ریشه‌دار شده T در u باشد. ما همچنین درخت ریشه‌دار شده‌ی T^u شامل v و تمام نسل‌های v را با T^u_v مشخص می‌کنیم.

برای پیداکردن (k,l)ـ هسته‌ی یک درخت مانند مقالات پنگ[۲۱] و لو[۲۲] [۵۱] و بکر [۸] الگوریتمی بازگشتی را با شروع از یک رأس مرکزی استفاده می‌کنیم. یک رأس مرکزی v از درخت T مرکز درخت بدون وزن متناظر با T است و رأسی است که بیشترین تعداد رئوس زیردرخت‌های به دست آمده از حذف آن را کمینه می‌کند. بعد از پیداکردن رأس مرکزی v، الگوریتم، یک (k,l)ـ هسته شامل v در درخت T^v را پیدا می‌کند. اگر آن (k,l)ـ هسته‌ی درخت نباشد، آن‌گاه (k,l)ـ هسته باید به طور کامل در یکی از زیردرخت‌های ریشه‌دار شده در رئوس همسایه‌ی v قرار داشته باشد. الگوریتم برای این زیردرخت‌ها به طور بازگشتی اجرا می‌شود.

نکته این که اگر $w(T)<0$ آن‌گاه مطابق با قضیه‌ی ۲.۳.۲ (k,l)ـ هسته یک رأس است. بنابراین الگوریتم باید یک رأس را به عنوان (k,l)ـ هسته پیدا کند. همچنین

[۲۱] *Peng*
[۲۲] *Lo*

در این حالت مطابق با لم ۴.۳.۲ رأس مورد نظر یا برگ است و یا یک رأس داخلی u به طوری که برای هر رأس v همسایه با u $w(u) \geq 0$ و $w(T_{uv}) \leq 0$. بنابراین در الگوریتم بعد از پیداکردن یک رأس مرکزی اگر رأس مرکزی یک برگ نباشد و $w(u) \geq 0$، زیردرخت‌های همسایه با آن باید بررسی شوند. اگر همه‌ی آن‌ها وزن منفی داشته باشند رأس مرکزی (k,l)ـ هسته نیز است. در غیر اینصورت رأس مرکزی حذف و زیردرخت‌های به دست آمده از حذف آن یکی یکی بررسی می‌شوند.

قبل از معرفی الگوریتم مفاهیم زیر را از [۸] بیان می‌کنیم.

فرض کنید $f(u)$ پدر u باشد. فاصله‌ی ذخیره شده‌ی $sav(v, P_{vu})$ به دست آمده از اضافه‌کردن مسیر P_{vu} به ریشه توسط رابطه‌ی زیر به دست می‌آید:

$$sav(v, P_{vu}) = sav(v, P_{vf(u)}) + a(f(u), u)sum_c(u)$$

که اگر $v = f(u)$ آن گاه $sav(v, P_{vf(u)}) = sav(v, v) = 0$ و $sum_c(u)$ به صورت زیر محاسبه می شود.

$$sum_c(v) = \begin{cases} w(v) & \text{اگر } v \text{ همسایه‌ی } T_c \text{ باشد} \\ w(v) + \sum_{u \ a \ child \ of \ v} sum_c(u) & \text{درغیراینصورت} \end{cases} \quad (۲.۲)$$

فرض کنید $P = P_{vu}$ مسیر داده شده باشد و B مجموعه فرزندان v باشد. زیردرختی که P در آن قرار دارد را توسط $T_{\bar{b}}^v$ با $b \neq \bar{b}$ نشان می‌دهیم.

در الگوریتم، درخت T^v به صورت زیر هرس می‌شود:

الگوریتم هرس

۱. برای به دست آوردن مسیرهای با طول حداکثر $min\{l-L(P), L(P)\}$، مسیرهایی را که متعلق به زیردرخت‌های T_b^v هستند را با $b \neq \bar{b}$ هرس کنید.

۲. برای مسیرهایی که در زیردرخت مشابه مانند $P = P_{vu}$(در غیر اینصورت در $T_{\bar{b}}^v$) قرار می‌گیرند، مسیرهای P_{xw} با $x \in P$، $x \neq v$ و $w \in T_{\bar{b}}^v \setminus P$ را هرس کنید به طوری که

$$L(P_{xw}) > min\{L(P) - L(P_{vx}), l - L(P_{xu})\}$$

درخت هرس شده را $\hat{T^v}$ می‌نامیم و وزن‌های درخت $\hat{T^v}$ به صورت زیر محاسبه می‌شوند:

$$w'(u) = \begin{cases} w(u) & u \text{ برگ } \hat{T^v} \text{ نیست} \\ sum_v(u)a(u, f(u)) & u \text{ برگ } \hat{T^v} \text{ است} \end{cases} \quad (۳.۲)$$

۱.۴.۲ الگوریتم

ورودی: یک درخت T با وزن مثبت و منفی.
خروجی: یک (k,l)ـ هسته مانند S^* از درخت T و DISTSUM آن d^*.
شروع
۱ـ قرار دهید $d^* := +\infty$.
۲ـ زیربرنامه‌ی SUBTREE(T) را اجرا کنید.
پایان

زیربرنامه $SUBTREE(T')$
ورودی: یک زیردرخت $T' = (V', E')$ از T با $| V' |= n'$ و بهترین DISTSUM برابر d^* داده شده.
خروجی: اگر بهترین زیردرخت در T'، DISTSUM کمتر از مقدار قبلی d^* داشته باشد، بهترین
زیردرخت S^* در T'، با حداکثر k برگ و با قطر حداکثر l و DISTSUM آن به عنوان
مقدار جدید d^*.
شروع
۱ـ **اگر** T' شامل یک رأس باشد **آن‌گاه** $S' = T'$ و $d(S')$ مقدار DISTSUM آن است.
در غیر اینصورت مراحل زیر را انجام دهید.
۱-۱ـ یک رأس مرکزی v از T' پیدا کنید.
۱-۲ـ **اگر** $w(T) < 0$ **آن‌گاه**
۱-۲-۱ـ **اگر** v یک رأس داخلی است و $w(v) \geq 0$ و برای هر رأس u همسایه با v، $w(T_{vu}) \leq 0$
آن گاه v، (k,l)ـ هسته است. قرار دهید $S' = v$ و DISTSUM آن $d(S')$ است.
در غیر اینصورت قرار دهید $S' :=$**BEST-TREE**(T', v).
۱-۳ـ **اگر** $d(S') < d^*$ **آن‌گاه** قرار دهید $d^* := d(S')$ و $S^* := S'$.
۱-۴ـ **برای هر** زیردرخت T^i به دست آمده از T' با حذف v زیربرنامه‌ی $SUBTREE(T^i)$ را اجرا
کنید.
پایان

زیربرنامه $BEST-TREE(T', v)$ (بکر [۸])

ورودی :یک زیردرخت $T' = (V', E')$ از T با n' رأس ریشه‌دار شده در رأس مرکزی v.
خروجی: بهترین زیردرخت S' شامل v در T'^v که حداکثر k برگ دارد و قطر آن حداکثر l و DISTSUM آن $d(S')$ است.
شروع
مسیرهایی که از v شروع می‌شوند و طول آن‌ها حداکثر l است را پیدا کنید.
برای هر مسیر P که از v شروع می‌شود و طول آن حداکثر l .است
مراحل زیر را انجام دهید.
۱ـ درخت T'^v را هرس کنید و فرض کنید $\hat{T'^v}$ درخت جدید باشد (الگوریتم هرس را ببینید.)
۲ـ فاصله ذخیره شده‌ی مسیرهایی از v به هر رأس $u \neq v$ در $\hat{T'^v}$ را پیدا کنید.
۳ـ **اگر** $deg(v) \geq ۲$ **آن‌گاه** مسیر P' را پیدا کنید. قرار دهید $P'' = P \cup P'$.
در غیر اینصورت
قرار دهید $P'' = P$ و مجموعه‌ی LS را با توجه به P'' پیداکنید.
۴ـ **اگر** $\hat{T'^v}$ بیشتر از k برگ داشته باشد **آن‌گاه**
$k - ۲$ مسیر را در LS پیدا کنید تا به P'' اضافه شود.
فرض کنید S' بهترین زیردرخت شامل v و $d(S')$، DISTSUM آن باشد.
در غیر اینصورت
همه‌ی مسیرها در LS را انتخاب کنید.
قرار دهید $S' = \hat{T'^v}$ و DISTSUM آن را برابر $d(S')$ قرار دهید.
پایان

قضیه ۱.۴.۲. مسأله‌ی پیداکردن یک $(k, l)-$هسته‌ی درخت T با وزن مثبت و منفی می‌تواند در زمان $O(n^۲ logn)$ حل شود.

برهان. اگر $w(T) < ۰$ آن‌گاه مطابق با قضیه‌ی ۲.۳.۲ (k, l)ـهسته یک رأس است. بنابراین الگوریتم باید یک رأس را به عنوان (k, l)ـهسته پیدا کند. همچنین در این حالت مطابق با لم ۴.۳.۲ رأس مورد نظر یا برگ است و یا یک رأس داخلی u به طوری که برای هر رأس v همسایه با u $w(u) \geq ۰$ و $w(T_{uv}) \leq ۰$. بنابراین در الگوریتم بعد از پیداکردن یک رأس مرکزی اگر رأس مرکزی یک برگ نباشد و $w(u) \geq ۰$، زیردرخت‌های همسایه با آن باید بررسی شوند. اگر همه‌ی آن‌ها وزن منفی داشته باشند رأس مرکزی (k, l)ـهسته نیز است. در غیر اینصورت رأس مرکزی حذف و زیردرخت‌های به دست آمده از حذف آن یکی یکی بررسی می‌شوند. بنابراین پیچیدگی این الگوریتم مانند الگوریتم بکر [۸] $O(n^۲ logn)$ است. □

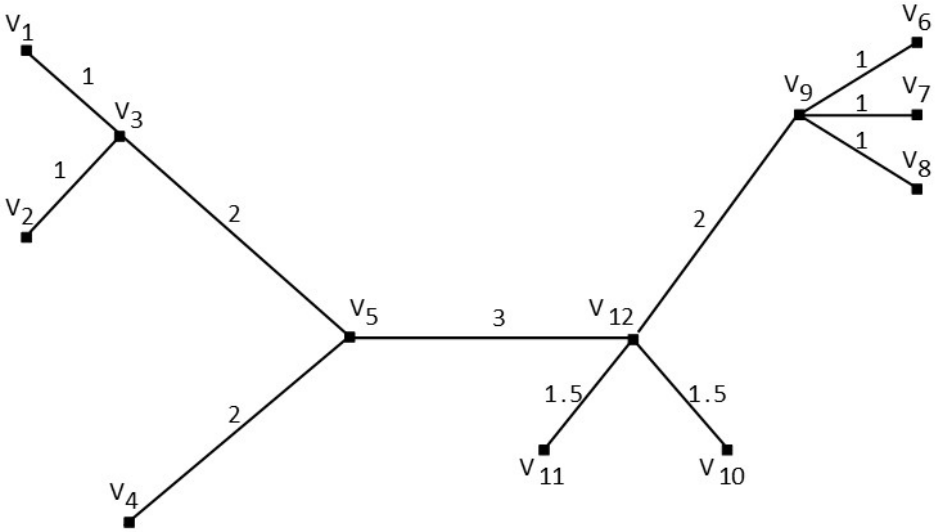
۵.۲ مثال های عددی

در این بخش برای حالات مختلف، الگوریتم را به کار می‌بریم. در مثال ۱.۵.۲، ۲.۵.۲ و ۳.۵.۲ به ترتیب وزن کل رئوس درخت مثبت، منفی و صفر است.

مثال ۱.۵.۲. درخت نمایش داده شده در شکل ۴.۲ را در نظر بگیرید. طول یال‌ها روی آن‌ها نوشته شده و وزن رئوس در جدول ۱.۲ نشان داده شده است. وزن کل درخت $w(T) = ۱ > ۰$ است. فرض کنید می‌خواهیم $(۳,۴)$ـهسته را روی این درخت پیدا کنیم. طبق الگوریتم باید یک رأس مرکزی پیدا کنیم. رأس $v_{۱۲}$ را به عنوان رأس مرکزی در نظر می‌گیریم. سپس باید همه‌ی مسیرهایی که از $v_{۱۲}$ شروع می‌شوند و طول آن ها حداکثر ۴ است را پیدا کنیم. این مسیرها در شکل ۵.۲ نشان داده می‌شوند.

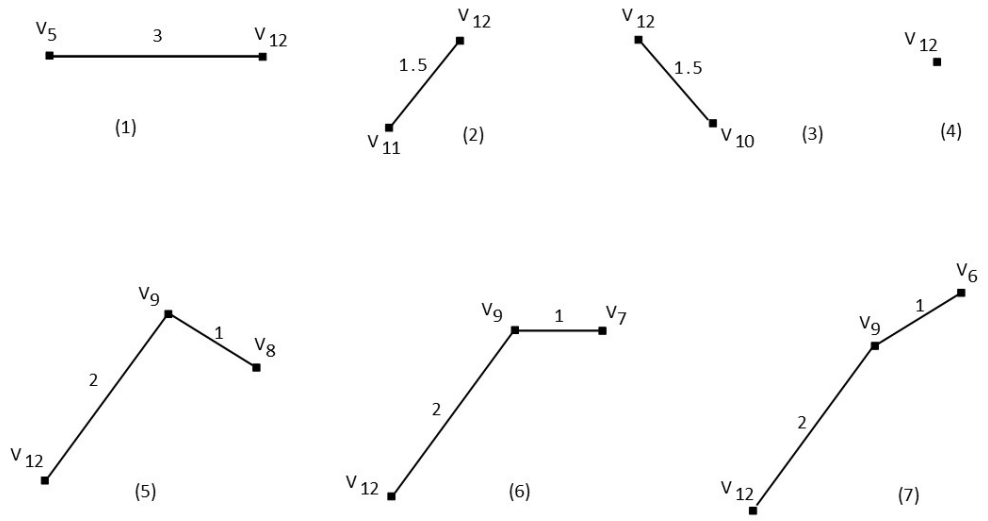
$w_{۱۲}$	$w_{۱۱}$	$w_{۱۰}$	$w_۹$	$w_۸$	$w_۷$	$w_۶$	$w_۵$	$w_۴$	$w_۳$	$w_۲$	$w_۱$
۱	۲	−۳	−۲	۱	−۲	−۱	۳	۱	۴	−۱	−۲

جدول ۱.۲: وزن‌های رئوس درخت در شکل ۴.۲ برای مثال ۱.۵.۲.



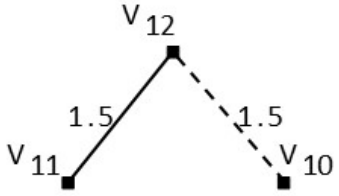
شکل ۴.۲: یک درخت با ۱۲ رأس.

اکنون برای هر مسیر درخت T را هرس می‌کنیم و گام‌های الگوریتم را ادامه می‌دهیم. برای مثال مسیر شماره ۲ را درنظر می‌گیریم. درخت جدید بعد از هرس در



شکل ۵.۲: مسیرهایی که از $v_{۱۲}$ شروع می‌شوند و طول آن ها حداکثر ۴ است.

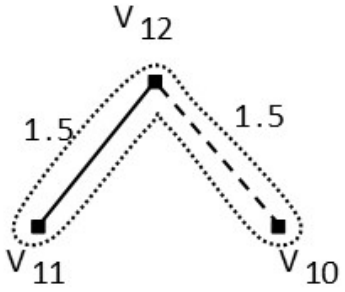
شکل ۶.۲ نشان داده شده است.



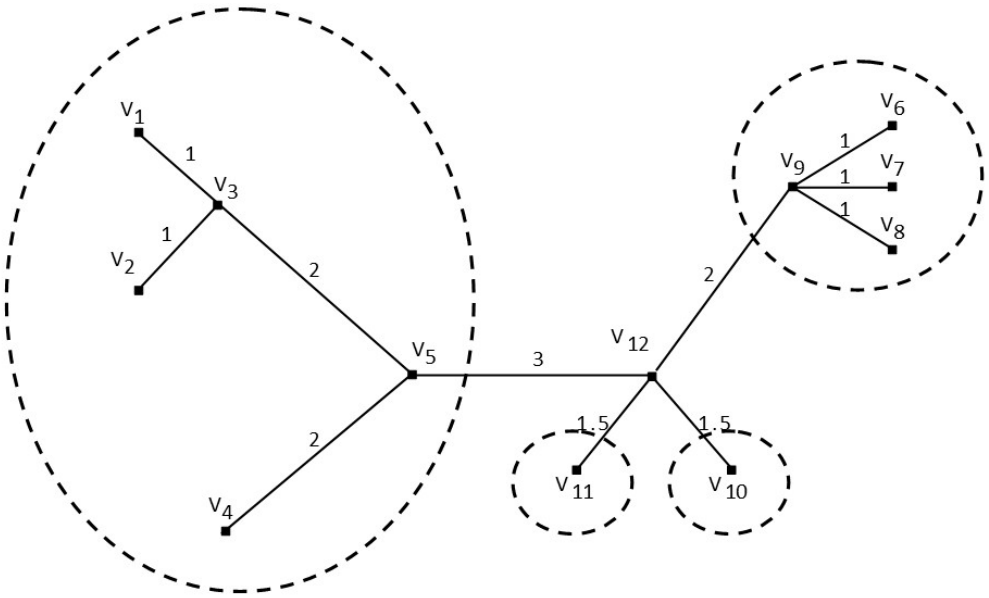
شکل ۶.۲: مسیر شماره ۲ بعد از هرس.

فاصله‌ی ذخیره شده‌ی $p : v_{۱۲}\_v_{۱۰}$ برابر با $sav(v_{۱۲}, p_{v_{۱۲}v_{۱۰}}) = -۴/۵$ و فاصله‌ی ذخیره شده‌ی $p : v_{۱۲}\_v_{۱۱}$ برابر با $sav(v_{۱۲}, p_{v_{۱۲}v_{۱۱}}) = +۳$ است. از آن جایی که $d(v_{۱۲}) = ۲$ است، $p' = v_{۱۲} - v_{۱۰}$ را در نظر می‌گیریم.
بنابراین $p'' = p \cup p'$ که در شکل ۷.۲ نشان داده شده است. همچنین از آن جایی که $T^{\wedge' v}$ ، ۲ برگ دارد، طبق الگوریتم همه‌ی مسیرها در LS را انتخاب می‌کنیم. در نتیجه $S' = T^{\wedge' v}$ و $d(S') = ۶$. بعد از اجرای گام‌های فوق برای هر مسیر، زیردرخت‌های به دست آمده از حذف $v_{۱۲}$ را یکی یکی درنظر می‌گیریم (شکل ۸.۲ را ببینید) و یک رأس مرکزی برای هر زیردرخت پیدا کنید. گام‌های فوق را برای هر زیردرخت ادامه می‌دهیم.

در نهایت زیردرختی که در شکل ۹.۲ نشان داده شده است، $(۳,۴)$ـ هسته است و $distsum$ آن برابر با $-۲۶/۵$ می باشد.



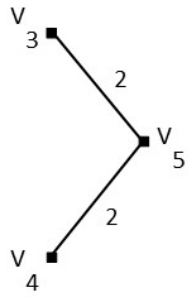
شکل ۷.۲: مسیر p''.



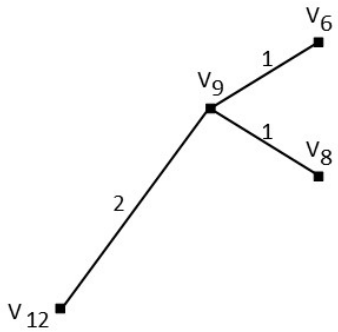
شکل ۸.۲: زیردرخت‌های به دست آمده از حذف $v_{۱۲}$.

مثال ۲.۵.۲. اکنون شکل ۴.۲ را با وزن‌های رئوس نشان داده شده در جدول ۲.۲ در نظر می‌گیریم. در این مثال داریم $w(T) = -۱ < ۰$. بنابراین $(۳,۴)$ـ هسته باید یک رأس باشد. رأس $v_۴$، $(۳,۴)$ـ هسته است و $d(v_۴) = -۳۳/۵$.

مثال ۳.۵.۲. دوباره شکل ۴.۲ را درنظرمی‌گیریم. وزن‌های رئوس در جدول ۳.۲ نشان داده شده است. مجموع وزن‌های همه‌ی رئوس در این مثال $w(T) = ۰$ است. $(۳,۴)$ـ هسته در شکل ۱۰.۲ ارائه شده است. $distsum$ آن برابر با -۱۲ است. نکته این که با وجود این که وزن درخت صفر است $(۳,۴)$ـ هسته یک رأس نیست.



شکل ۹.۲: $(۳,۴)$ـ هسته .



شکل ۱۰.۲: $(۳,۴)$ـ هسته .

۶.۲ نتیجه‌گیری

در این فصل مسأله‌ی پیدا کردن (k,l)ـ هسته‌ی یک درخت با وزن مثبت و منفی را بررسی کردیم. با استفاده از قضایای ارائه شده در [۶۳] نشان دادیم، زمانی که وزن درخت منفی باشد، جواب مسأله‌ی (k,l)ـ هسته‌ی یک درخت با وزن مثبت/منفی همان جواب مسأله‌ی ۱−میانه است. سپس بر مبنای قضایای ارائه شده، الگوریتمی با پیچیدگی زمانی $O(n^۲logn)$ که در واقع اصلاحی از الگوریتم بکر[۸] است، برای حل مسأله‌ی (k,l)ـ هسته‌ی یک درخت با وزن مثبت و منفی ارائه دادیم. در مطالعات بعدی می‌توان معکوس مسأله‌ی هسته را حل کرد، به این صورت که هسته‌ی یک درخت معلوم و مکان مشتری‌ها مجهول است. به عبارت دیگر هدف در این مسأله، پیداکردن مکان مشتری‌ها است، به طوری که مجموع فاصله‌ی مشتری‌ها تا هسته

$w_{۱}$	$w_{۲}$	$w_{۳}$	$w_{۴}$	$w_{۵}$	$w_{۶}$	$w_{۷}$	$w_{۸}$	$w_{۹}$	$w_{۱۰}$	$w_{۱۱}$	$w_{۱۲}$
-۲	-۲	۳	۱	۳	-۲	-۳	۲	۱	-۱	-۲	۱

جدول ۲.۲: وزن‌های رئوس درخت در شکل ۴.۲ برای مثال ۲.۵.۲.

$w_{۱}$	$w_{۲}$	$w_{۳}$	$w_{۴}$	$w_{۵}$	$w_{۶}$	$w_{۷}$	$w_{۸}$	$w_{۹}$	$w_{۱۰}$	$w_{۱۱}$	$w_{۱۲}$
-۱	۱	-۱	-۱	۲	۱	۱	۱	-۳	۱	۱	-۲

جدول ۳.۲: وزن‌های رئوس درخت در شکل ۴.۲ برای مثال ۳.۵.۲.

کمینه شود. همچنین می‌توان این مساله را به مسأله‌ی (k,l)ـ هسته نیز تعمیم داد.

فصل ۳

الگوریتم ژنتیک و مسأله‌ی $(k,l)-$هسته

۱.۳ مقدمه

در این فصل مسأله‌ی پیداکردن $(k,l)-$هسته‌ی یک شبکه را با وزن مثبت و منفی(نیمه‌ـ ناخوشایند) در نظر می‌گیریم. این مسأله یک مسأله‌ی $NP-$سخت است. بنابراین برای حل آن از روش‌های ابتکاری و فراابتکاری استفاده می‌شود. در کنار روش‌های جستجوی فراابتکاری[۱]، روش‌های جستجوی دیگری هم وجود دارند که به الگوریتم‌ـ های تکاملی[۲] معروفند. بر اساس نظریه‌ی داروین نسل‌هایی که از ویژگی‌ها و خصوصیات برتری نسبت به نسل‌های دیگر برخوردارند شانس بیشتری برای بقا و تکثیر خواهند داشت و ویژگی‌ها و خصوصیات برتر آن‌ها به نسل‌های بعدی آن‌ها نیز منتقل خواهد شد. همچنین بخش دوم نظریه‌ی داروین بیان می‌کند که هنگام تکثیر یک فرزند، به تصادف رویدادهایی اتفاق می‌افتد که موجب تغییر خصوصیات فرزند بعد از جهش می‌شود و در صورتی که این تغییر فایده‌ای برای فرزند جدید داشته باشد موجب افزایش احتمال بقای این فرزند خواهد شد. در محاسبات کامپیوتری، بر اساس نظریه‌ی

---

[۱] *Meta heuristic*

[۲] *Evolutionary algoritms*

داروین روش‌هایی برای مسائل بهینه‌سازی ارائه شده است که همه‌ی این روش‌ها از پردازش تکاملی در طبیعت نشأت گرفته‌اند. به این روش‌های جستجو، الگوریتم‌های جستجوی تکاملی می‌گوییم.
الگوریتم‌های تکاملی مانند الگوریتم‌های فراابتکاری، به شکل تصادفی اما هدایت‌شده در فضای نمونه‌ی مسأله حرکت می‌کنند. با توجه به نظریه‌ی داروین، پیاده‌سازی یک الگوریتم تکاملی طی مراحل زیر انجام می‌پذیرد[۵۷]:

۱. کدگذاری[۳] و نحوه‌ی نمایش ،

۲. تولید جمعیت اولیه ،

۳. محاسبه‌ی میزان بهینگی هر یک از افراد جمعیت(تابع شایستگی[۴])،

۴. انتخاب،

۵. تولید نسل [۵]،

۶. جهش[۶] .

الگوریتم‌های ژنتیک هم به عنوان یکی از ابزارهای قدرتمند در حل مسائل بهینه‌سازی از مراحل مذکور برای رسیدن به جواب استفاده می‌کنند.
ما برای حل مسأله‌ی $(k,l)-$هسته سه الگوریتم ژنتیک ارائه می‌دهیم. الگوریتم ژنتیک یک روش جستجوی ابتکاری برای حل مسائل بهینه‌سازی ترکیبیاتی است. این الگوریتم با یک جواب شدنی شروع شده و با بهبوددادن این جواب ادامه می‌یابد. الگوریتم ژنتیک در سال ۱۹۶۰ توسط جان هلند[۷] معرفی شد. اما در سال‌های اخیر مورد توجه محققان تحقیق‌درعملیات قرار گرفته است. جزئیات بیشتر در مورد الگوریتم ژنتیک در کتاب‌های گلدبرگ[۸] [۲۹] و ریوز[۹] [۵۳] آمده است. الگوریتم‌های ژنتیک زیادی برای حل برخی از مسائل مکانیابی از قبیل مسائل مکانیابی میانه و مسیر طراحی شده‌اند. اخیراً رهبری و فتحعلی [۵۲] ترکیبی از الگوریتم ژنتیک و مورچگان را برای پیداکردن مسیر مرکزی یک گراف ارائه دادند. در ادامه ابتدا الگوریتم ژنتیک را در حالت کلی معرفی کرده و سپس مدل ژنتیک مسأله‌ی $(k,l)-$هسته آورده شده است. بعد از آن سه الگوریتم ژنتیک برای پیداکردن $(k,l)-$هسته‌ی یک شبکه در حالت نیمه‌ناخوشایند

---

[۳] *Encoding*
[۴] *Fitness*
[۵] *Reproduction*
[۶] *Mutation*
[۷] *John Holland*
[۸] *Goldberg*
[۹] *Reeves*

ارائه شده است. در انتها نتایج محاسباتی این سه الگوریتم و مقایسه‌ی آن‌ها با هم آورده شده‌اند.

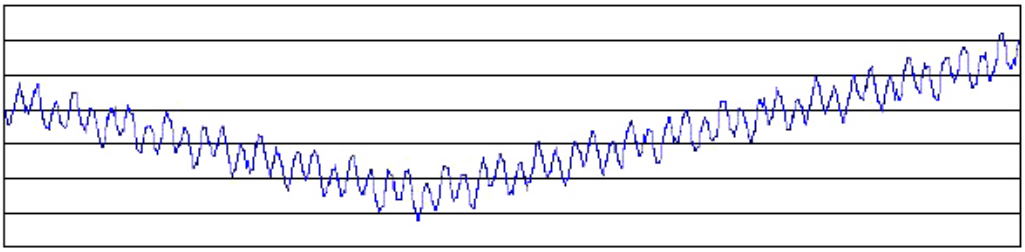
۲.۳ الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک، روش‌های فراابتکاری هستند که با الهام از طبیعت برای حل مسائل بر مبنای سیستم تکاملی و ژنتیک‌های طبیعت به کار برده می‌شوند. این الگوریتم‌ها با تولید یک جمعیت اولیه شروع می‌شوند. افراد این جمعیت کروموزوم‌ها هستند که توسط طراح الگوریتم کدگذاری شده‌اند. هر فرد یا هر کروموزوم در جمعیت یک جواب مسأله است. این جواب‌های اولیه توسط عملگرهای ادغام و جهش بهبود پیدا می‌کنند. یک کروموزوم از تعدادی ژن تشکیل شده است. این ژن‌ها خصوصیات یک فرد را مشخص می‌کنند. برای ارزیابی کروموزوم‌ها از معیاری به نام مقدار شایستگی[۱۰] استفاده می‌شود. جمعیت اولیه معمولاً به صورت تصادفی تولید می‌شود. عملگر انتخاب، افراد بهتر را از جمعیت برای به وجود آوردن افراد جدید انتخاب می‌کند. احتمال انتخاب‌شدن هر کروموزوم بستگی به میزان شایستگی آن دارد. افرادی که مقدار شایستگی آن‌ها بالاتر باشد شانس بیشتری برای انتخاب شدن دارند. عملگرهای ادغام و جهش برای بهبود جواب‌ها و تولید افراد جدید استفاده می‌شوند. عملگر ادغام با ترکیب والدین کروموزوم‌های جدید را تولید می‌کند. برای وجود افراد جدید و متفاوت از اعضای جمعیت، عملگر جهش به تصادف برخی از ژن‌های کروموزوم‌ها را تغییر می‌دهد. جهش باید طوری طراحی شود که از همگرایی زودرس الگوریتم ژنتیک به جواب‌هایی که بهینه نیستند جلوگیری کند. به عبارت دیگر عملگر جهش مانع از افتادن الگوریتم در تله‌ی بهینه‌ی موضعی می‌شود. فرآیند تولید افراد جدید و جایگزینی جمعیت تا برآورده‌شدن شرط توقف ادامه می‌یابد.
اکنون قسمت‌های مختلف یک الگوریتم ژنتیک در حالت کلی بیان می‌شود.

۱.۲.۳ فضای جستجو

وقتی به دنبال جواب مسأله‌ای هستیم، بهترین جواب‌ها را در مجموعه جواب‌های شدنی جستجو می‌کنیم. فضای تمام جواب‌های شدنی، فضای جستجو نامیده می‌شود. هر نقطه در فضای جستجو یک جواب شدنی را نشان می‌دهد. هر جواب شدنی می‌تواند براساس ارزش و مطلوبیتش برای مسأله درجه‌بندی شود. بهترین جواب از بین مجموعه جواب‌های شدنی با یک نقطه در فضای جستجو نشان داده می‌شود. شکل ۱.۳ یک نمونه از فضای جواب را نشان می‌دهد.

[۱۰] *Fitness value*



شکل ۱.۳: نمونه‌ای از فضای جواب

مشکلی که در این‌جا وجود دارد این است که جستجو می‌تواند خیلی پیچیده باشد. مشخص نیست که کجا باید به دنبال جواب باشیم و از کجا باید شروع کنیم. روش‌های زیادی برای پیداکردن جواب‌های مناسب وجود دارد. به عنوان مثال می‌توان به الگوریتم‌های تکاملی، جستجوی ممنوع (TS)[۱۱]، شبیه‌سازی تبرید(SA)[۱۲] و ... اشاره کرد.

۲.۲.۳ مشخصات بیولوژیکی

بدن تمام موجودات زنده از سلول تشکیل شده است. در هر سلول مجموعه‌ای از موجودات هم‌شکل به نام کروموزوم[۱۳] وجود دارند. کروموزوم‌ها، رشته‌های DNA هستند که هر کدام متشکل از تعدادی ژن[۱۴] هستند. هر ژن یک خصیصه را کد می‌کند. به عنوان مثال رنگ چشم می‌تواند یک خصیصه باشد. مجموعه‌های ممکن برای هر خصیصه آلل[۱۵] نامیده می‌شوند. هر ژن در کروموزوم موقعیت خاص خودش را دارد که این موقعیت خاص لوکوس[۱۶] نام دارد. مجموعه‌ی کامل همه‌ی کروموزوم‌ها، ژنوم[۱۷] نامیده می‌شود.

۳.۲.۳ کدگذاری و نحوه‌ی نمایش

الگوریتم ژنتیک به جای این که بر روی پارامترها یا متغیرهای مسأله کار کند، با شکل کدشده‌ی آن‌ها سروکار دارد. به عنوان مثال در مسأله‌ی ۸ وزیر، یک روش کدگذاری

[۱۱] *Tabu search*
[۱۲] *Simulated annealing*
[۱۳] *Chromosome*
[۱۴] *Gene*
[۱۵] *Allel*
[۱۶] *Locous*
[۱۷] *Genome*

می‌تواند استفاده از یک آرایه‌ی ۸ عنصری باشد که هر عنصر از آرایه نشان‌دهنده‌ی سطری است که وزیر در آن قرار دارد. به عنوان مثال آرایه‌ی زیر را در نظر بگیرید:

۴	۳	۲	۶	۸	۷	۵	۱

شکل ۲.۳: نمونه‌ی کدگذاری مسأله‌ی ۸ وزیر

با توجه به مقادیر آرایه در می‌یابیم که وزیر اول در ستون اول و سطر چهارم، وزیر دوم در ستون دوم و سطر سوم، وزیر سوم در ستون سوم وسطر دوم و ... از صفحه‌ی شطرنج قرار گرفته‌است. در الگوریتم‌های ژنتیک به هر جواب مسأله یک کروموزوم و به هر متغیر از آن یک ژن می‌گوییم و اندازه‌ی یک کروموزوم تعداد ژن‌های آن است. در این‌جا جواب مسأله‌ی هشت وزیر را با استفاده از یک آرایه‌ی ۸ عنصری نشان دادیم که به آرایه یک کروموزوم و به هر عنصر از آرایه یک ژن می‌گوییم. با توجه به مسأله‌ی بهینه‌سازی برای هر ژن می‌توان سؤالات زیر را مطرح کرد:

- آیا مقادیر قابل قبول برای ژن در یک بازه‌ی مشخص قرار دارد؟
- آیا دامنه‌ی مقادیر ژن گسسته است یا پیوسته؟
- آیا در صورت مسأله، محدودیت‌هایی برای انتخاب مقادیر ژن وجود دارد؟

نحوه‌ی نمایش جواب مسأله و ژن‌ها در موفقیت الگوریتم و نحوه‌ی پیاده‌سازی الگوریتم ژنتیک تأثیر بسیار مهمی دارد.

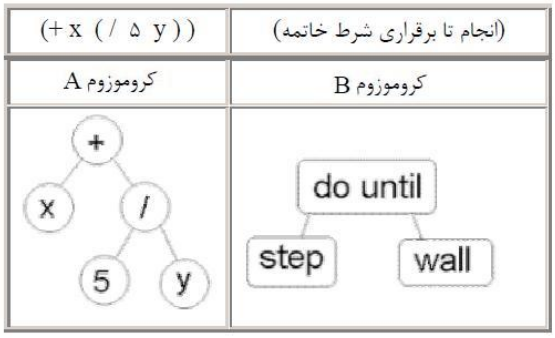
نمایش باینری

درکدگذاری باینری[۱۸] هر ژن از کروموزوم مقدار ۰ یا ۱ می‌گیرد و موجب تشکیل یک رشته‌ی باینری در کروموزوم می‌گردد. به عنوان مثال، فرض کنید یک تابع سه متغیره داریم و می‌خواهیم مقادیر این سه متغیر را طوری انتخاب کنیم که مقدار این تابع مینیمم گردد. فرض کنید هر متغیر یک مقدار اعشاری ۴ بایتی(۳۲ بیتی) باشد. در صورتی که بخواهیم از روش نمایش باینری برای این سه متغیر استفاده کنیم، باید یک رشته باینری به طول ۹۶ بیت تشکیل دهیم.

[۱۸] *Binary encoding*

نمایش درختی

کدگذاری درختی[۱۹] در برنامه‌های تکاملی به منظور برنامه‌ریزی تکاملی بکار می‌رود. در کدگذاری درختی هر کروموزوم یک درخت از اشیائی نظیر توابع یا دستورها در زبان برنامه‌نویسی است. شکل زیر دو نمونه از این کروموزوم‌ها را نشان می‌دهد. زبان برنامه‌نویسی $LISP$ اغلب از این کدگذاری استفاده می‌کند. زیرا برنامه‌های آن به این فرم نمایش داده می‌شوند و می‌توانند به راحتی مورد تجزیه قرار بگیرند.



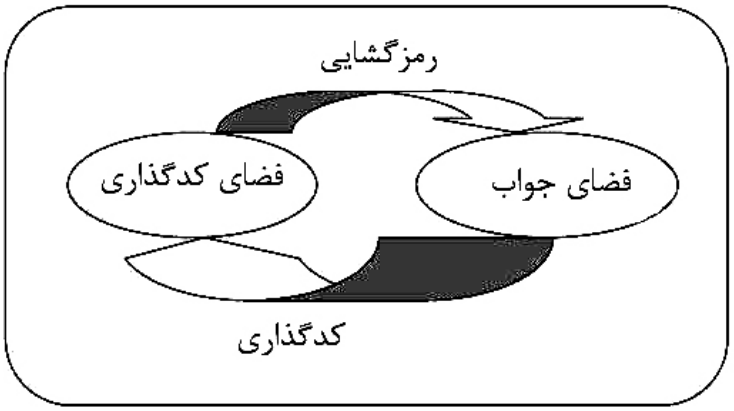
شکل ۳.۳: کدگذاری درختی

نکته‌ای که در انتهای این قسمت باید به آن توجه کرد این است که در الگوریتم‌های ژنتیک، کدگذاری، یک رابطه بین فضای کدگذاری و فضای جواب‌ها است، به طوری‌که الگوریتم‌های ژنتیک عملیات تکاملی را به طور متناوب در این دو فضا انجام می‌دهند(شکل ۴.۳).

۴.۲.۳ تولید جمعیت اولیه

الگوریتم ژنتیک کار خود را با تولید جمعیت اولیه‌ای از کروموزوم‌ها آغاز می‌کند و سپس در یک حلقه به طور مکرر تعدادی از کروموزوم‌های برتر نسل فعلی را انتخاب کرده و سپس نسل جدیدی از این کروموزوم‌ها را تولید می‌کند. همان‌طور که دیدیم هر کروموزوم نشان‌دهنده‌ی یک عنصر از فضای نمونه‌ی مسأله است و بنابراین منظور از تولید جمعیت اولیه، تولید تعدادی جواب برای مسأله خواهد بود. تولید جواب‌های اولیه نیز به دو صورت تصادفی و ابتکاری می‌تواند انجام پذیرد.
به عنوان مثال در مسأله‌ی ۸ وزیر می‌توانیم به شکل تصادفی وزیرها را در خانه‌های صفحه‌ی شطرنج قراردهیم و یا در مسأله‌ی فروشنده‌ی دوره‌گرد می‌توانیم ترتیب

[۱۹] *Tree encoding*



شکل ۴.۳: فضای کدگذاری و فضای جواب

ملاقات شهرها را هم به شکل تصادفی هم به شکل ابتکاری انتخاب کنیم.

نکته‌ی مهم هنگام تولید جمعیت اولیه آن است که هنگام تولید جواب برای مسأله(تصادفی یا ابتکاری)، جواب تولید شده باید به شکل کروموزومی که در مرحله‌ی قبل تعیین کردیم، کدگذاری شده و به مجموعه جمعیت اضافه شود. بنابراین باید از تولید جواب‌ـ هایی که موجب نقض نحوه‌ی کدگذاری تعیین شده برای کروموزوم شود، اجتناب کنیم.

حال ممکن است این سؤال مطرح شود که تعداد کروموزوم‌هایی که برای جمعیت اولیه تولید می‌کنیم چگونه تعیین می‌شود؟ جواب قطعی برای این سؤال وجود ندارد و در برخی موارد تعداد افراد جمعیت(کروموزوم‌های) به شکل تجربی انتخاب می‌شود. اما یک اصل کلی وجود دارد که در تعیین اندازه‌ی جمعیت اولیه باید مدنظر قرار گیرد. اگرچه افزایش اندازه‌ی جمعیت موجب افزایش قدرت محاسباتی الگوریتم می‌شود، اما به طور چشم‌گیری مدت زمان اجرای الگوریتم را افزایش می‌دهد. همچنین در برخی موارد بزرگی جمعیت موجب توقف زودرس الگوریتم می‌شود که منجر به تولید جواب های بهینه‌ی محلی خواهد شد، در حالی که هدف پیداکردن بهینه‌ی سراسری است. در مقابل، جمعیت با اندازه‌ی کوچک نیز توانایی چندانی به خصوص در مسائل پیچیده برای حل مسأله نخواهد داشت.

۵.۲.۳ تابع شایستگی

الگوریتم ژنتیک نیز مانند الگوریتم‌های فراابتکاری در هر بار تکرار به تدریج خود را به نقطه‌ی بهینه‌ی سراسری مسأله نزدیک می‌کند. همچنین گذر از نمونه‌ای به نمونه‌ی دیگر براساس هزینه‌ی هر نمونه انجام می‌پذیرد. در روش‌های جستجوی فراابتکاری تخمین هزینه‌ی هر نمونه با استفاده از تابع هزینه انجام می‌گیرد. به عنوان مثال هزینه‌ی هر نمونه در مسأله‌ی ۸ وزیر برابر است با تعداد گاردهایی که وزیرها به

همدیگر می‌دهند. در الگوریتم ژنتیک نیز می‌توان مستقیماً از تابع هزینه برای تخمین شایستگی یک نمونه استفاده کرد.
در واقع تابع شایستگی میزان برتری یک نمونه نسبت به نمونه‌ی دیگر را تخمین می‌زند و موجب می‌شود که هنگام تکثیر، نمونه‌هایی به نسل بعد منتقل شوند که شایستگی بیشتری داشته باشند.

۶.۲.۳ عملگر انتخاب

با توجه به نظریه‌ی داروین برای تولید نسل جدید از جمعیت فعلی، باید کروموزوم‌هایی از این جمعیت را برای ادغام و تکثیر انتخاب کنیم که هنگام اجرای عمل انتخاب، کروموزوم‌هایی که شایستگی بیشتری دارند، شانس بیشتری برای انتخاب داشته باشند. کروموزوم‌های انتخاب شده برای ادغام با دیگر کروموزوم‌ها به منظور تولید نسل جدید در محل دیگری جمع‌آوری می‌شوند. عمل ادغام و تولید کروموزوم‌های جدید در این محل که آن را حوضچه‌ی ژنتیک می‌نامیم، انجام می‌پذیرد. انتخاب کروموزوم‌ها از جمعیت فعلی هم به روش‌های گوناگونی انجام می پذیرد که ما در این‌جا متداول‌ترین روش‌های انتخاب را بررسی می‌کنیم.
مسأله‌ی دیگر در انتخاب کروموزوم‌ها اندازه‌ی حوضچه‌ی ژنتیک است. بدین معنی که چه تعداد کروموزوم برای ادغام و تولید نسل جدید انتخاب کنیم؟ روش معمول آن است که اندازه‌ی حوضچه دقیقاً به همان اندازه‌ی جمعیت فعلی باشد. ممکن است چنین تصور شود که انتخاب حوضچه‌ی ژنتیک به اندازه‌ی جمعیت فعلی بدین معنا است که تمام کروموزوم‌های جمعیت فعلی را به حوضچه‌ی ژنتیک منتقل کنیم. در حالی که چنین نیست و هنگام انتخاب کروموزوم‌ها، آن‌هایی که از شایستگی کمتری برخوردارند به این حوضچه منتقل نمی‌شوند و در مقابل کروموزوم‌های بهتر ممکن است چند بار در آن کپی شوند. بر اساس نظریه‌ی داروین هنگام تکثیر کروموزوم‌ها، خصوصیات برتر هر کروموزوم پس از ادغام با دیگر کروموزوم‌ها به نسل بعدی منتقل شده و موجب تولید کروموزوم‌های بهتری می‌شود. با این تعریف، در صورتی که چند کپی از یک کروموزوم بهینه در حوضچه‌ی ژنتیک وجود داشته باشد، موجب می‌شود که در تولید نسل بعدی از کروموزوم‌های بهتر نسل فعلی بیشتر استفاده شده و بهینگی نسل بعدی بهتر از نسل فعلی باشد. با وجود این که این نظریه در طبیعت صادق است، اما در برخی موارد این نظریه هنگام پیاده‌سازی الگوریتم ژنتیک با شکست مواجه می شود. با این وجود در بیشتر موارد نتایج بسیار جالبی را به همراه دارد. در نتیجه کروموزوم‌های با شایستگی کم نیز باید در حوضچه‌ی ژنتیک حضور داشته باشند.
روش‌های دیگری نیز در انتخاب اندازه‌ی حوضچه‌ی ژنتیک وجود دارد. به عنوان مثال می‌توان اندازه‌ی حوضچه را دو برابر جمعیت فعلی درنظر گرفت. همچنین می‌توان در هر بار تولید نسل جدید اندازه‌ی حوضچه را تغییر داد و روش‌های دیگری که یک

طراح الگوریتم ژنتیک می‌تواند طراحی کند. نکته‌ی مهم، نحوه‌ی انتخاب کروموزوم‌ها از جمعیت فعلی و قرار دادن آن‌ها در حوضچه‌ی ژنتیک است. روش‌های مختلفی برای انتخاب کروموزوم‌ها و ادغام آن‌ها وجود دارند که مهم‌ترین این روش‌ها عبارتند از :

- روش تصادفی[۲۰]،
- روش رقابتی[۲۱]،
- روش چرخ رولت[۲۲]،
- روش رتبه‌ای[۲۳]،
- روش بولتزمن[۲۴]،
- روش حالت پایدار[۲۵].

۷.۲.۳ عملگر ادغام

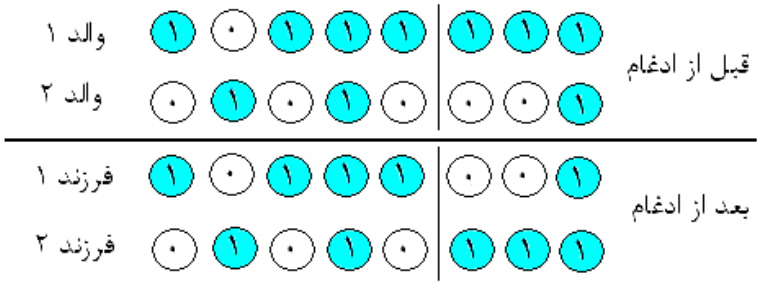
پس از انتخاب کروموزوم‌ها و انتقال آن‌ها به حوضچه، دو مرحله‌ی دیگر نیز نمودی از نظریه‌ی داروین هستند. این دو مرحله عبارتند از ادغام و جهش که در ادامه به بررسی آن‌ها می‌پردازیم.
بخشی از فرآیند تکامل در طبیعت بدین صورت است که کروموزوم‌هایی از والدین انتخاب شده و با هم ترکیب می‌شوند. ما نیز در مرحله‌ی انتخاب، کروموزوم‌هایی را برای ترکیب به حوضچه‌ی ژنتیک منتقل کردیم. حال باید کروموزوم‌هایی را از این حوضچه انتخاب کرده و با هم ادغام کنیم. به همین منظور در این بخش روش‌هایی را برای ادغام کروموزوم‌ها بررسی خواهیم کرد. توجه شود که در برخی از مسائل و با توجه به روش کدگذاری کروموزوم‌ها، عمل ادغام کروموزوم‌ها موجب نقض برخی از شرایط مسأله می‌گردد. بنابراین در چنین مواردی باید با استفاده از روش‌هایی که طراحی می‌کنیم به اصلاح کروموزوم‌ها بپردازیم.

ادغام تک نقطه‌ای

در ادغام تک نقطه‌ای[۲۶]، نقطه‌ای از کروموزوم را به تصادف انتخاب کرده و سپس برای تولید دو کروموزوم فرزند، ژن‌های بعد از این نقطه را در دو والد با هم جابه‌جا

[۲۰] *Random*
[۲۱] *Tournament*
[۲۲] *Roulette wheel*
[۲۳] *Rank*
[۲۴] *Boltzman*
[۲۵] *Steady state*
[۲۶] *Single – sight crossover*

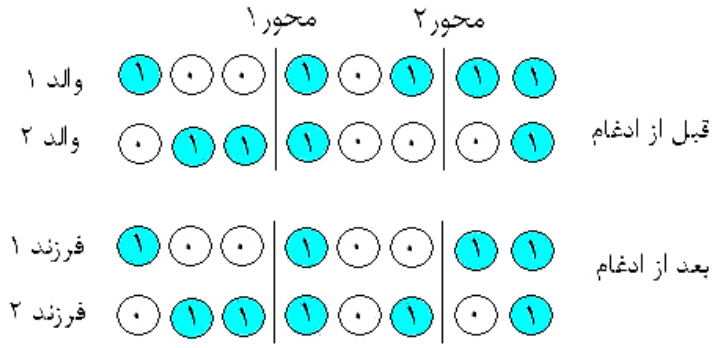
می‌کنیم. نقطه‌ی انتخاب شده را محور[۲۷] می‌نامیم(شکل ۵.۳).



شکل ۵.۳: ادغام تک نقطه‌ای

ادغام دونقطه‌ای

در ادغام دو نقطه‌ای[۲۸]، دو نقطه‌ی متفاوت از کروموزوم را به تصادف انتخاب کرده و سپس به روش زیر مقادیر بین این دونقطه جابه‌جا و دو کروموزوم فرزند تولید می‌شود. دو نقطه‌ی انتخاب شده را محور۱ و محور۲ می‌نامیم (شکل ۶.۳).



شکل ۶.۳: ادغام دو نقطه‌ای

ادغام چند نقطه‌ای

درادغام چند نقطه‌ای[۲۹]، m نقطه به صورت تصادفی و بدون تکرار تعیین شده و به صورت صعودی مرتب می‌شوند. سپس متغیرهای بین هر دو نقطه یکی در میان بین دو والد جا به جا می‌شوند تا به این ترتیب دو فرزند جدید تولید شوند. روند کلی این ادغام در شکل‌های ۷.۳ و ۸.۳ آمده است.

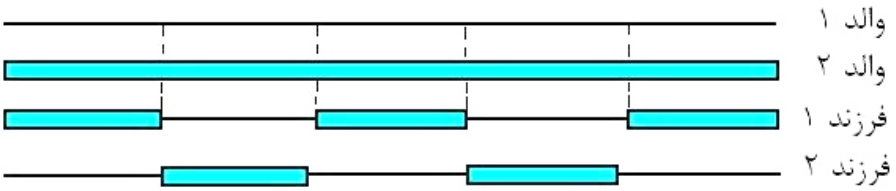
---

[۲۷] $Pivot$

[۲۸] $Two-point\ crossover$

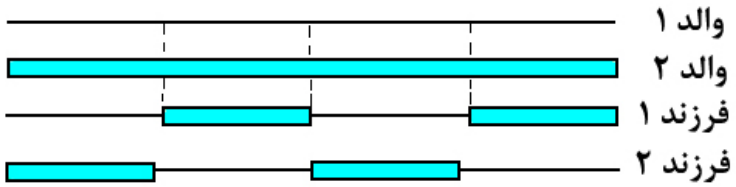
[۲۹] $Multi-point\ crossover$

در حالتی که تعداد نقاط زوج باشد



شکل ۷.۳: ادغام چند نقطه‌ای در حالتی که تعداد نقاط زوج باشد

در حالتی که تعداد نقاط فرد باشد



شکل ۸.۳: ادغام چند نقطه‌ای در حالتی که تعداد نقاط فرد باشد

مشاهده می‌شود هنگامی که تعداد نقاط زوج است، قسمت‌های اول و آخر کروموزوم‌های فرزند، از یک والد مشترک ارث برده‌اند.

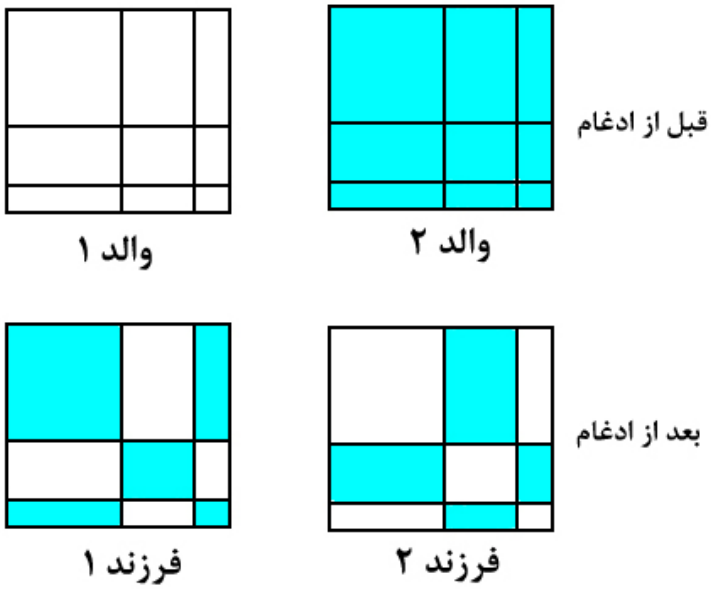
ادغام ماتریسی

در ادغام ماتریسی[۳۰]، برای نشان دادن کروموزوم‌ها از ماتریس استفاده می‌کنیم به طوری که دو موقعیت تصادفی در سطر و ستون ماتریس انتخاب می‌شود. عمل ادغام در شکل ۹.۳ نشان داده شده است.

۸.۲.۳ عملگر جهش

نظریه‌ی داروین بیان می‌کند که پس از تولید کروموزوم‌های فرزند، ممکن است در برخی از این کروموزوم‌ها جهش‌هایی به تصادف روی دهند که موجب بهترشدن کروموزوم و یا بدترشدن آن گردند. ما نیز در استفاده از الگوریتم‌های ژنتیک از این خصوصیت استفاده کرده و جهش‌هایی را در جواب‌های تولید شده برای مسأله ایجاد

[۳۰] *Matrix crossover*



شکل ۹.۳: ادغام ماتریسی

می‌کنیم. این روش نیز بسته به ابتکار طراح الگوریتم به روش‌های گوناگونی پیاده‌سازی می‌شود. اما روش کلی کار بدین شکل است که مقدار یک یا چند عدد از ژن‌های یک کروموزوم تغییر داده می‌شوند. برای این منظور می‌توان دو ژن را به تصادف انتخاب کرده و جای آن‌ها را با هم تعویض کرد، یا می‌توان ژنی را انتخاب کرد و به تصادف مقدار جدیدی را به آن ژن نسبت داد. همچنین می‌توان ژنی را به تصادف انتخاب کرده و مکمل آن را به عنوان مقدار جدید ژن در نظر گرفت. به عنوان مثال در مسأله‌ی ۸ وزیر می‌توان وزیری را به تصادف انتخاب کرده و مکان آن را در ستون مربوط به خود به شکل تصادفی تغییر داد. پس از اعمال مراحل ادغام و جهش کار تولید نسل جدید به پایان می‌رسد و در این مرحله باید عمل جایگزینی بین نسل جدید و جمعیت فعلی انجام پذیرد.

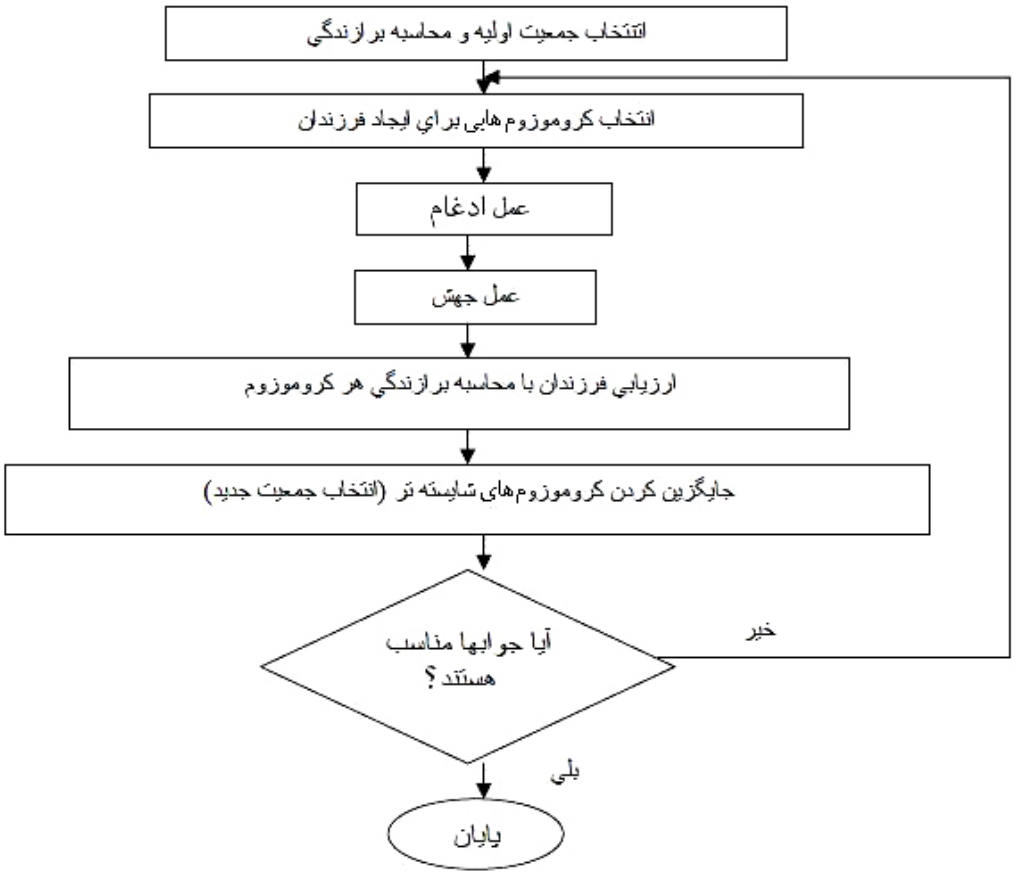
۹.۲.۳ سیاست جایگزینی

جایگزینی جمعیت فعلی با نسل جدید در حالت کلی به دو روش انجام می‌پذیرد.

- همه‌ی کروموزوم‌های نسل جدید را جایگزین کروموزوم‌ها در جمعیت فعلی کنیم(در حالتی که اندازه‌ی نسل جدید برابر با جمعیت فعلی باشد، کل جمعیت فعلی با نسل جدید جایگزین می‌شود).

- برخی از کروموزوم‌های جمعیت فعلی را انتخاب کرده و آن‌ها را با کروموزوم دیگری در نسل جدید جایگزین کنیم.

مراحل اجرای الگوریتم ژنتیک، در حالت کلی در شکل ۱۰.۳ نشان داده شده است.
در بخش‌های بعد سه الگوریتم ژنتیک برای پیدا کردن $(k,l)-$هسته‌ی یک شبکه ارائه



شکل ۱۰.۳: مراحل اجرای الگوریتم ژنتیک

می‌دهیم. آن‌ها را $GKLC$۱، $GKLC$۲ و $GKLC$۳ نام‌گذاری کرده‌ایم. در ادامه ابتدا این الگوریتم‌ها را توضیح داده و سپس آن‌ها را از نظر کیفیت و سرعت رسیدن به جواب با هم مقایسه می‌کنیم. تمامی این الگوریتم‌ها به زبان متلب نوشته شده‌اند.

۳.۳ الگوریتم $GKLC$۱

در این بخش ساختار الگوریتم ژنتیک را برای مسأله‌ی پیدا کردن $(k,l)-$هسته‌ی یک شبکه بیان می‌کنیم.

۱.۳.۳ کد گذاری

در یک الگوریتم ژنتیک هر کروموزوم متناظر با یک جواب مسأله است و هر جواب در مسأله‌ی ما یک درخت می‌باشد. ما برای هر جواب یک ماتریس مربعی با بعد n در نظر می‌گیریم. در این ماتریس سطرها و ستون‌ها متناظر با رئوس هستند. اگر یالی بین دو رأس v_i و v_j در این جواب وجود داشته باشد درایه‌ی (i,j) از ماتریس ۱ است. در غیر اینصورت بی‌نهایت می‌باشد. عناصر روی قطر اصلی صفر هستند.

۲.۳.۳ مقدار شایستگی

مقدار شایستگی یک کروموزوم ، مقدار تابع هدف جواب متناظر برای مسأله‌ی $(k,l)-$هسته است.

۳.۳.۳ جمعیت اولیه

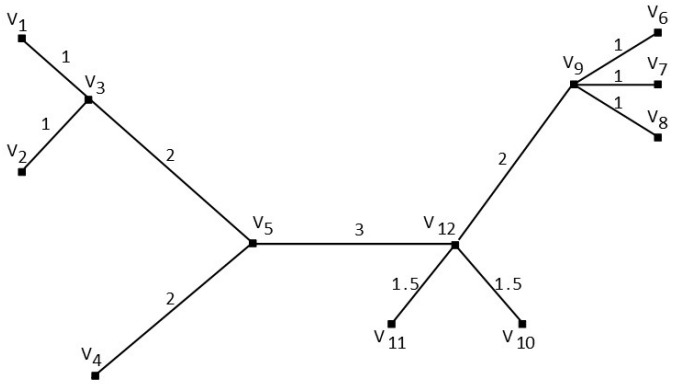
جمعیت اولیه در الگوریتم‌های ژنتیک بر همگرایی الگوریتم تأثیر دارد. یک جمعیت اولیه‌ی مناسب منجر به همگرایی سریع الگوریتم می‌شود. اندازه‌ی جمعیت در یک الگوریتم ژنتیک به نوع و اندازه‌ی مسأله بستگی دارد. ما در الگوریتم‌هایمان اندازه‌ی جمعیت را با NP نشان می‌دهیم.
برای تولید هر عضو از جمعیت اولیه NP رأسی که بیشترین وزن را در گراف دارند انتخاب می‌کنیم. هر یک از این رئوس اولین رأس از یک عضو جمعیت اولیه است. بنابراین برای ساختن هر یک از افراد جمعیت اولیه از یکی از این NP رأس انتخاب شده شروع می‌کنیم. فرض کنید v_i یکی از رئوس انتخاب شده باشد. برای ساختن یک درخت ما از $T = v_i$ شروع کرده و مقدار شایستگی یا $F(T)$ را محاسبه می‌کنیم. سپس از بین همسایگان v_i رأس v_j با بیشترین وزن را پیدا می‌کنیم. فرض کنید P، مسیر از v_i به v_j باشد. اگر $f(P) < f(T)$ آن گاه v_j به درخت T اضافه می‌شود. با روش مشابه مسیر را ادامه می‌دهیم تا زمانی که طول آن کمتر از l باشد یا مقدار تابع هدف با اضافه کردن رأس جدید به مسیر موجود نتواند کاهش پیدا کند. بعد از ساخته‌شدن مسیر با طول کمتر از l برای داشتن یک درخت می‌توان برگ‌هایی را به این مسیر اضافه کرد. به این منظور همسایه‌های رئوس داخلی مسیر با بیشترین وزن را به T اضافه می‌کنیم. این عملیات را ادامه می‌دهیم تا زمانی که تعداد برگ‌های درخت تولید شده کمتر از k باشد یا مقدار تابع هدف با اضافه‌کردن برگ جدید به درخت موجود نتواند کاهش پیدا کند(زیربرنامه‌ی جمعیت اولیه را ببینید).

زیربرنامه‌ی جمعیت اولیه‌ی (T)

ورودی: گراف G با وزن مثبت/ منفی.
خروجی: مجموعه‌ی S شامل زیردرخت‌های G.
شروع
۱ـ **قرار دهید** $S := \emptyset$.
۲ـ **تا زمانی که** تعداد اعضای S بیشتر از اندازه‌ی جمعیت نشده است، مراحل زیر را انجام دهید.
۲-۱ـ رأس $v \in V$ با بیشترین وزن را که قبلاً انتخاب نشده است، انتخاب کنید.
و قرار دهید،
$f_{best} := f(v)$،
$S(v) := v$،
$P(v) := v$ و
$EndP := v$
۲-۲ـ **تا زمانی که** مجموعه‌ی Adj۱ $:= \{u \in V | u \notin P(v),$ همسایه‌ی $EndP$ است $u\}$ تهی نیست و $d_{P(v)} \leq l$
مراحل زیر را انجام دهید.
۲-۲-۱ـ یک رأس $u \in Adj$۱ که بیشترین وزن را دارد انتخاب کنید.
۲-۲-۲ـ رأس u را به $P(v)$ اضافه کنید و قرار دهید $EndP := u$.
۲-۲-۳ـ **اگر** $f(P(v)) < f_{best}$ **آن‌گاه** $f_{best} := f(P(v))$ و $S(v) := P(v)$.
۲-۳ـ K را تعداد برگ های $P(v)$ قرار دهید و تا وقتی که مجموعه‌ی Adj۲ $:= \{u \in V | u \notin P(v),$ همسایه‌ی یک رأس داخلی $P(v)$ است $u\}$ تهی نیست و $K \leq k$
مراحل زیر را انجام دهید.
۲-۳-۱ـ رأس $u \in Adj$۲ را که دارای بیشترین وزن است انتخاب کنید و u را به $P(v)$ اضافه کنید و $K := K + ۱$.
۲-۳-۲ـ **اگر** $f(p) < f_{best}$ **آن‌گاه** $f_{best} := f(p)$ و $S(v) := P(v)$.
۲-۴ـ $S := S \cup \{S(v)\}$.
پایان

برای توضیح بیشتر این الگوریتم مثال زیر را مشاهده کنید.

مثال ۱.۳.۳. گراف نشان داده شده در شکل ۱۱.۳ را در نظر بگیرید. وزن رئوس این درخت در جدول ۱.۳ نشان داده شده است.

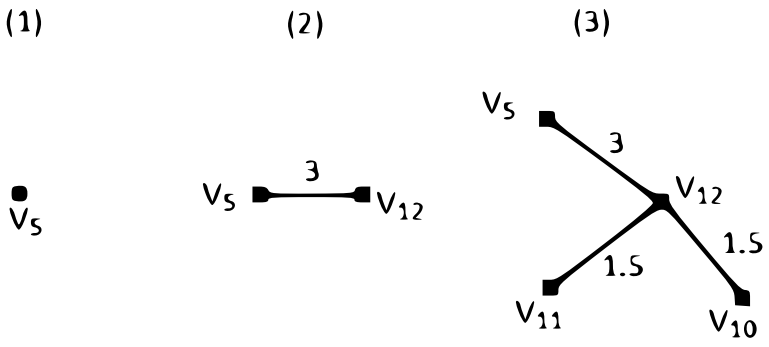


شکل ۱۱.۳: .یک گراف با ۱۲ رأس

$w_{۱۲}$	$w_{۱۱}$	$w_{۱۰}$	$w_۹$	$w_۸$	$w_۷$	$w_۶$	$w_۵$	$w_۴$	$w_۳$	$w_۲$	$w_۱$
۱	۱	۱	-۳	۱	۱	۱	۲	-۱	-۱	۱	-۱

جدول ۱.۳: وزن‌های رئوس شکل ۱۱.۳.

برای تولید اولین عضو جمعیت، رأس با بالاترین وزن را انتخاب می‌کنیم. رأس $v_۵$ بالاترین وزن را در بین رئوس دارد. بنابراین از این رأس شروع می‌کنیم. شماره ۱ را در شکل ۱۲.۳ ببینید. سپس رأس $v_{۱۲}$ را که بالاترین وزن را در بین همسایه‌های رأس $v_۵$ دارد به آن اضافه می‌کنیم. شماره ۲ را در شکل ۱۲.۳ ببینید. فرض کنید مسأله‌ی ما $(۳,۴/۵)-$هسته باشد. بنابراین رئوس $v_{۱۰}$ و $v_{۱۱}$ را نیز به رأس $v_۵$ اضافه می‌کنیم. شماره ۳ را در شکل ۱۲.۳ ببینید.



شکل ۱۲.۳: مراحل تولید یک عضو از جمعیت اولیه.

۴.۳.۳ انتخاب

برای تولید اعضای جدید از بین افراد جمعیت، فرد با بیشترین مقدار تابع هدف را به عنوان والد اول انتخاب کرده و والد دوم را به تصادف انتخاب می‌کنیم.

۵.۳.۳ ادغام

در یک الگوریتم ژنتیک والدین انتخاب شده توسط عملگر ادغام با هم ترکیب می‌شوند تا فرزندان تولید شوند. ما در این الگوریتم ابتدا یال‌های مشترک دو والد را در کروموزوم‌های فرزندان یعنی فرزند۱ و فرزند۲ قرار می‌دهیم. سپس برای تولید کروموزوم فرزند که یک درخت متصل است، باید قسمت‌های غیرمتصل به هم وصل شوند. برای این کار یال‌هایی را به آن‌ها اضافه می‌کنیم. فرض کنید u و v دو رأس غیرمتصل در فرزند۱ و فرزند۲ باشد و $P۱_{uv}$ و $P۲_{uv}$ به ترتیب مسیرهای بین u و v در والد۱ و والد۲ باشند. در گام اول ما $P۱_{uv}$ و $P۲_{uv}$ را به ترتیب به فرزند۱ و فرزند۲ اضافه می کنیم. سپس برای دو رأس غیرمتصل بعدی در فرزند۱ و فرزند۲ ترتیب تخصیص مسیرها را تغییر می‌دهیم. به عبارت دیگر یک قسمت از والد اول و والد دوم به ترتیب به فرزند۲ و فرزند۱ اضافه می‌کنیم. این روش را ادامه می‌دهیم تا زمانی که دو مسیر متصل در فرزند۱ و فرزند۲ تولید شوند. سپس با روشی مشابه تولید جمعیت اولیه، رئوسی را به رئوس انتهایی ورئوس داخلی هر فرزند اضافه می‌کنیم. برای هر فرزند ابتدا همسایه با بالاترین وزن به رئوس انتهایی اضافه و سپس برگ‌ها به رئوس داخلی اضافه می‌شوند. این عملیات ادامه می‌یابد تا زمانی که تعداد برگ‌های درخت تولیدشده به k برسد یا نتوانیم رئوس جدید را به آن اضافه کنیم. این روش تولید افراد جدید در زیربرنامه‌ی ادغام نشان داده شده است.

زیربرنامه ی ادغام $(T_۱, T_۲)$

ورودی: دو زیردرخت $T_۱$ و $T_۲$ از G.
خروجی: زیردرخت $T_{crossover}$ از G.
شروع
۱ـ یال‌های مشترک $T_۱$ و $T_۲$ را در $T_{new۱}$ و $T_{new۲}$ قرار دهید.
۲ـ **قرار دهید** i=۱.
۳ـ **برای** هر دو رأس u و v که باعث یک انفصال در $T_{new۱}$ و $T_{new۲}$ هستند،
مراحل زیر را انجام دهید.
۱-۳ـ **اگر** i فرد باشد
۱-۱-۳ـ مسیر $P۱_{uv}$ را از u به v در $T_۱$ و مسیر $P۲_{uv}$ را در $T_۲$ پیدا کنید.
۲-۱-۳ـ مسیر $P۱_{uv}$ را به $T_{new۱}$ و $P۲_{uv}$ را به $T_{new۲}$ اضافه کنید.

۳-۱-۳ـ قرار دهید $i = i + ۱$.

در غیر اینصورت اگر i زوج باشد

۳-۱-۴ـ مسیر $P۱_{uv}$ را از u به v در $T_۲$ و مسیر $P۲_{uv}$ را در $T_۱$ پیدا کنید.

۳-۱-۵ـ مسیر $P۱_{uv}$ را به $T_{new۱}$ و مسیر $P۲_{uv}$ را به $T_{new۲}$ اضافه کنید.

۳-۱-۶ـ **قرار دهید** $i = i + ۱$.

۳-۲ـ **قرار دهید**

$f_{crossover۱} := f(T_{new۱})$ و $T_{crossover۱} := T_{new۱}$

$f_{crossover۲} := f(T_{new۲})$ و $T_{crossover۲} := T_{new۲}$

۳-۳ـ **برای** $T_{new} = T_{new۱}$ و $T_{new۲}$

مراحل زیر را انجام دهید.

۳-۳-۱ـ **فرض کنید** x رأس پایانی T_{new} باشد.

۳-۳-۲ـ **تا زمانی که مجموعه‌ی**

$Adj۳ := \{y \in T_۱ \cup T_۲ \setminus T_{new}, y$ همسایه‌ی x است$\}$ تهی نیست و $d_{T_{new}} \leq l$

مراحل زیر را انجام دهید.

۳-۳-۲-۱ـ رأس $y \in Adj۳$ را با بیشترین وزن انتخاب کنید.

۳-۳-۲-۲ـ **اگر** با اضافه کردن y به T_{new}، درخت T_{new} شامل هیچ دوری نبود،

y را به انتهای T_{new} اضافه کنید.

۳-۳-۲-۳ـ **اگر** $f(T_{new}) < f_{crossover۱}$ **آن‌گاه**

$f_{crossover۱} := f(T_{new})$ و $T_{crossover۱} := T_{new}$.

۳-۳-۲-۴ـ **قرار دهید** $x = y$.

۳-۳-۳ـ K را تعداد برگ های T_{new} قرار دهید و تا وقتی که مجموعه‌ی

$Adj۴ := \{t \in V | t \notin T_{new}, t$ همسایه‌ی یک رأس داخلی T_{new} است$\}$

تهی نیست و $K \leq k$

مراحل زیر را انجام دهید.

۳-۳-۳-۱ـ رأس $t \in Adj۴$ را با بیشترین وزن انتخاب کنید.

۳-۳-۳-۲ـ **اگر** با اضافه کردن t به T_{new}، درخت T_{new} شامل هیچ دوری نبود،

t را به T_{new} اضافه کنید و قرار دهید $K := K + ۱$.

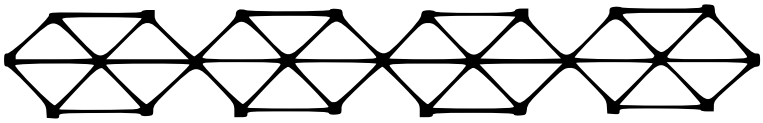
۳-۳-۳-۳ـ **اگر** $f(T_{new}) < f_{crossover۱}$ **آن‌گاه**

$f_{crossover۱} := f(T_{new})$ و $T_{crossover۱} := T_{new}$.

پایان

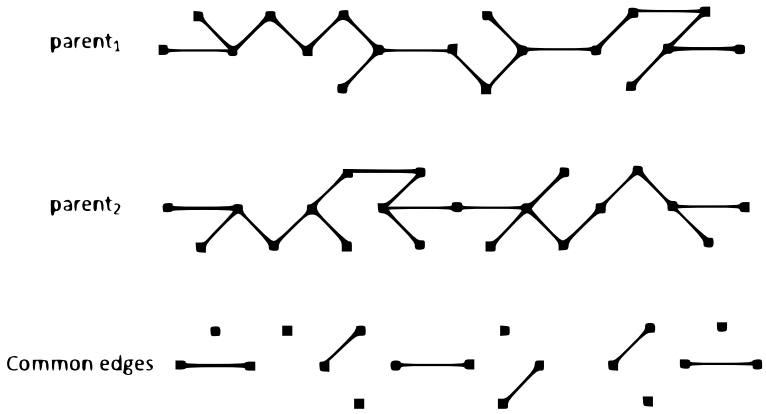
برای توضیح بیشتر این الگوریتم مثال زیر را داریم.

مثال ۲.۳.۳. گراف نشان داده شده در شکل ۱۳.۳ را در نظر بگیرید.



شکل ۱۳.۳: یک گراف با ۲۵ رأس.

طبق الگوریتم دو عضو از جمعیت را انتخاب می‌کنیم. فرض کنید $parent$۱ یک عضو از جمعیت اولیه با بهترین مقدار تابع شایستگی به عنوان والد۱ است و $parent$۲ عضو دیگری از جمعیت به عنوان والد۲ است که آن را به تصادف انتخاب کرده‌ایم. والدین انتخاب شده و یال‌های مشترکشان در شکل ۱۴.۳ آورده شده‌اند. شکل ۱۵.۳



شکل ۱۴.۳: والدین انتخاب شده و یال‌های مشترکشان.

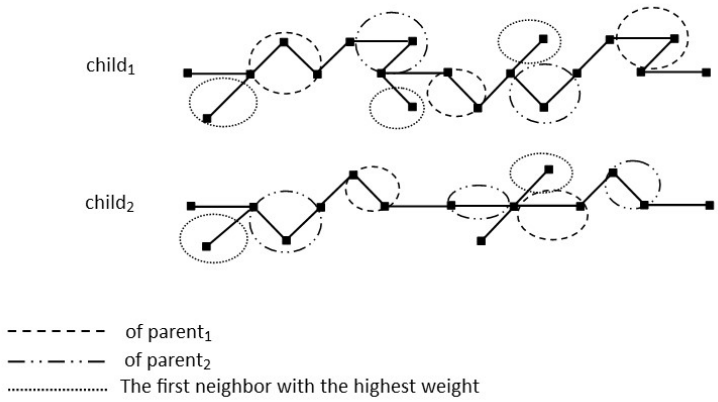
عملیات ادغام را در الگوریتم $GKLC$۱ نشان می‌دهد.

۶.۳.۳ جهش

عملگر جهش تنوع جمعیت را حفظ و از همگرایی زودرس الگوریتم جلوگیری می‌کند. برای به کارگیری این عملگر یک عضو از جمعیت را به تصادف انتخاب می‌کنیم. سپس هر برگ u را با رأس v که $v \notin T$ و بیشترین وزن را در بین همسایگان پدر u دارد جایگزین می‌کنیم(زیربرنامه جهش را ببینید).

زیربرنامه‌ی جهش T_m

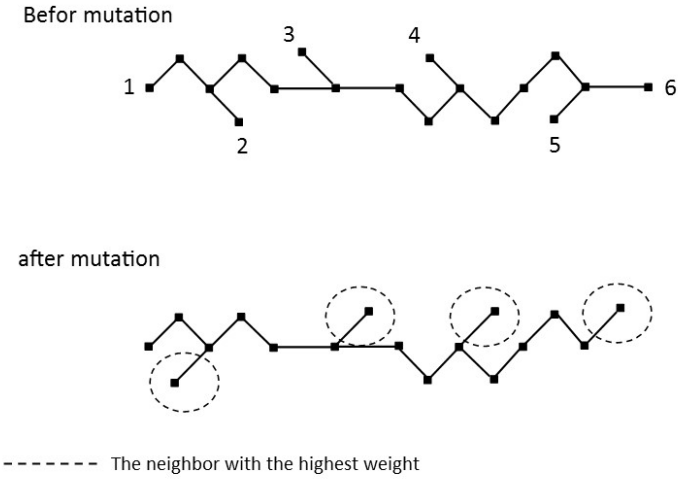
ورودی: یک زیردرخت T_m از G.
خروجی: یک زیردرخت $T_{mutation}$ از G.
شروع



شکل ۱۵.۳: عملیات ادغام در الگوریتم $GKLC$۱.

۱ـ **برای** هر برگ v از T_m
مراحل زیر را انجام دهید.
۱-۱ـ **قرار دهید** $Adj۵ := \{y \in G \setminus T_m \mid$ همسایه‌ی پدر v است $y\}$
۱-۲ـ رأس $y \in Adj۵$ را با بیشترین وزن انتخاب کنید.
۱-۳ـ v را توسط y جایگزین کنید.
۲ـ **قرار دهید** $T_{mutation} := T_m$.
پایان

مثال ۳.۳.۳. شکل ۱۶.۳ درخت T از شکل ۱۳.۳ را نشان می‌دهد. درخت قبل از جهش ۶ برگ دارد. طبق الگوریتم هر برگ را با رأس $v \notin T$ که بیشترین وزن را در بین همسایگان پدرش دارد، جایگزین می‌کنیم. از رأس ۱ شروع می‌کنیم. پدر رأس ۱ در شکل ۱۳.۳، ۳ همسایه دارد که همه‌ی آن‌ها در T هستند. مشاهده می‌شود رأسی که بتواند جایگزین رأس ۱ شود وجود ندارد. بنابراین رأس ۱ تغییر نمی‌کند. طبق الگوریتم، رئوس جدید جایگزین رئوس ۲، ۳ و ۴ می‌شوند. پدر رئوس ۵ و ۶ مشترک هستند و دو همسایه دارد که در T نیستند. طبق الگوریتم ما همسایه با بالاترین وزن را انتخاب می‌کنیم. بنابراین یک رأس جدید جایگزین دو رأس ۵ و ۶ می‌شود. شکل ۱۶.۳، T را بعد از جهش نشان می‌دهد.



شکل ۱۶.۳: عملگر جهش.

۷.۳.۳ جایگزینی

بعد از تولید کروموزوم‌های جدید توسط عملگر ادغام، آن‌ها را به جمعیت اضافه می‌کنیم. اگر عضو جدید با هیچ یک از اعضای موجود یکسان نبود و مقدار تابع شایستگی آن بهتر از مقدار تابع شایستگی بدترین عضو جمعیت بود، آن‌گاه عضو جدید جایگزین بدترین عضو می‌شود. این عمل را برای کروموزوم جدید تولیدشده توسط عملگر جهش تکرار می‌کنیم.

زیربرنامه‌ی جایگزینی

شروع

۱ــ **اگر** T_{new} در جمعیت نبود و $f(T_{new}) < f_{worst}$ **آن‌گاه** مراحل زیر را انجام دهید.

۱-۱ــ بدترین عضو جمعیت را با T_{new} جایگزین کنید.

۱-۲ــ بدترین عضو جمعیت و مقدار شایستگی f_{worst} آن به روزآوری کنید.

۱-۳ــ **اگر** $f(T_{new}) < f_{best}$ قرار بده $f_{best} = f(T_{new})$.

پایان

۸.۳.۳ شرط توقف

فرآیند فوق تا زمانی که به یک شرط توقف برسد، تکرار می‌شود. معمولاً شرط توقف یا تعداد تکرار الگوریتم است یا بیشترین تعداد تکرار بین دو بهبود و یا یک ماکزیمم زمان CPU است. ما در الگوریتم‌های ارائه شده، بیشترین تعداد تکرار را ۲۰ بار تکرار الگوریتم بعد از آخرین بهبود، یا در غیراینصورت رسیدن به تکرار صدم به عنوان شرط توقف در نظر می‌گیریم. البته در اکثر موارد قبل از رسیدن به تکرار صد، الگوریتم خاتمه پیدا می‌کند. بنابراین شرط توقف ۲۰ بار تکرار الگوریتم بعد از آخرین بهبود شرط توقف خوبی محسوب می‌شود.

۹.۳.۳ الگوریتم

مباحث ذکر شده در بخش‌های فوق، الگوریتم زیر را تشکیل می‌دهد.

الگوریتم ژنتیک $[GKLC۱]$
ورودی: یک گراف G با وزن مثبت/منفی.
خروجی: یک $(k,l)-$هسته‌ی S^* از G و $DISTSUM$ یا f_{best} آن.
شروع
جمعیت اولیه‌ی (T)
برای $t := ۰$ تا $t := ۲n$ **مراحل زیر را انجام دهید.**
۱-۱ـ $T_۱ \in S$ را با کمترین مقدار شایستگی و $T_۲ \in S$ را به تصادف انتخاب کنید.
۱-۲ـ **ادغام** $(T_۱, T_۲)$
۱-۳ـ **جایگزینی(**$T_{crossover۱}, S$**)**
۱-۴ـ **جایگزینی(**$T_{crossover۲}, S$**)**
۱-۵ـ درخت $T_m \in S$ را به تصادف انتخاب کنید.
۱-۶ـ **جهش(**T_m**)**
۱-۷ـ **جایگزینی(**$T_{mutaion}, S$**)**
۱-۸ـ زیردرخت T_f را در S با کمترین مقدار شایستگی پیدا کنید.
۱-۹ـ **قرار دهید**
$T_{best} := T_f$ و
$f_{best} := f(T_f)$ و
$t := t + ۱$
پایان

جمعیت اولیه و عملگرهای ادغام در سه الگوریتم با هم متفاوت هستند و بقیه‌ی عملگرها شبیه به هم عمل می‌کنند. جداول ۲.۳ و ۳.۳ این تفاوت‌ها را نشان می‌دهند.

جمعیت اولیه	الگوریتم‌ها	مزایا	معایب
با رئوسی که بیشترین وزن را دارند شروع می‌شوند	GKLC۱ و GKLC۲	پیداکردن جواب با کیفیت بالا	سرعت پایین
با رئوسی که به طور تصادفی انتخاب شده‌اند شروع می‌شود	GKLC۳	سرعت بالا	پیداکردن جواب با کیفیت پایین

جدول ۲.۳: تأثیر جمعیت اولیه در الگوریتم‌های ارائه شده.

ادغام	الگوریتم	مزایا	معایب
تمام یال‌های مشترک والدین یک به یک به فرزند اضافه می‌شوند	GKLC۱	پیداکردن جواب با کیفیت بالا	سرعت پایین
تمام یال‌های مشترک والدین دو به دو به فرزند اضافه می‌شوند	GKLC۲	پیداکردن جواب با کیفیت بالا	سرعت پایین
فقط یک یال مشترک از والدین درنظرگرفته می‌شود	GKLC۳	سرعت بالا	پیداکردن جواب با کیفیت پایین

جدول ۳.۳: تأثیر عملگر ادغام در الگوریتم‌های ارائه شده.

۴.۳ الگوریتم $GKLC$۲

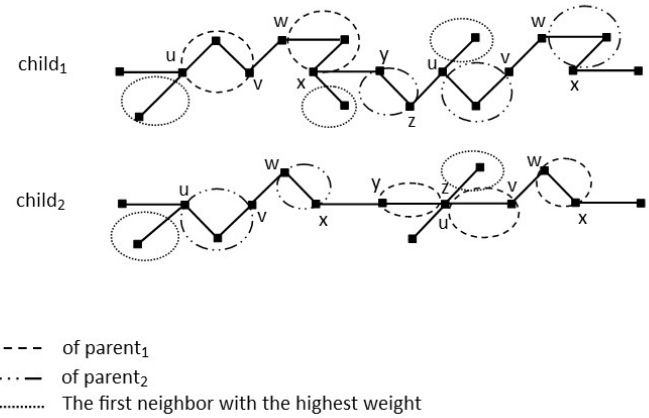
در این بخش عملگر ادغام الگوریتم $GKLC$۱ را تغییر می‌دهیم تا الگوریتم ژنتیک دیگری برای پیداکردن (k,l)−هسته‌ی یک شبکه معرفی کنیم. عملگر ادغام جدید در زیر توضیح داده شده است.

۱.۴.۳ ادغام

در این عملگر ادغام، مشابه الگوریتم $GKLC$۱ دو عضو از جمعیت اولیه را به عنوان دو والد عملگر ادغام مانند قبل انتخاب می‌کنیم. سپس یال‌های مشترک والدین را در دو فرزند۱ و فرزند۲ قرار می‌دهیم. اکنون برای متصل کردن یال‌های مشترک در هر فرزند باید یال‌هایی را به فرزندان اضافه کنیم. روش انتخاب این یال‌ها با آنچه که در الگوریتم $GKLC$۱ بیان کردیم متفاوت است. در این الگوریتم برای هر دو رأس غیرمتصل i و j در فرزند ۱ و ۲، فرض کنید P۱$_{ij}$ و P۲$_{ij}$ به ترتیب مسیرهای بین i و j در والد۱ و والد۲ باشند.

فرض کنید u، v، w x، y و z شش رأس غیرمتصل در فرزند۱ و فرزند۲ باشند. ابتدا P۱$_{uv}$، P۱$_{wx}$ و P۲$_{yz}$ را به فرزند ۱ و P۲$_{uv}$، P۲$_{wx}$ و P۱$_{yz}$ را به فرزند۲ اضافه می‌کنیم(شکل ۱۷.۳ را ببینید). سپس برای شش رأس غیرمتصل بعدی در فرزند۱ و فرزند۲ ترتیب تخصیص مسیرها را تغییر می‌دهیم. به عبارت دیگر دو قسمت از والد۲ و یک قسمت از والد۱ را به فرزند۱ و دو قسمت از والد۱ و یک قسمت از والد۲ را به

فرزند۲ اضافه می‌کنیم. بقیه‌ی الگوریتم مشابه با الگوریتم $GKLC$۱ می‌باشد.



شکل ۱۷.۳: عملیات ادغام در الگوریتم $GKLC$۲.

۵.۳ الگوریتم $GKLC$۳

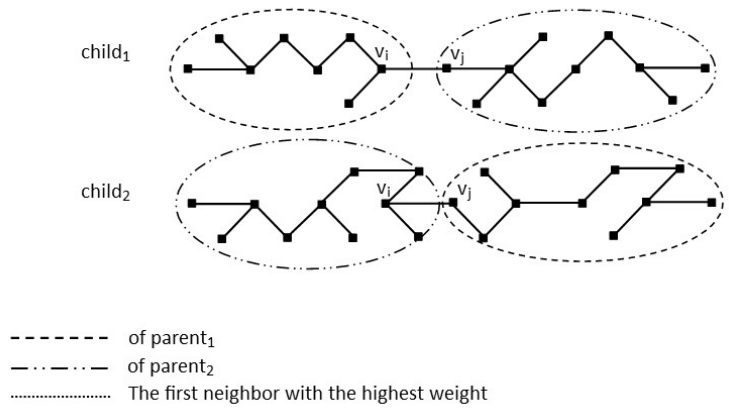
با کمی تغییر در دو الگوریتم قبل، الگوریتم ژنتیک دیگری برای پیداکردن $(k,l)-$هسته‌ی یک شبکه معرفی می‌کنیم. این الگوریتم مشابه با حالت قبل است به جز جمعیت اولیه و عملگر ادغام که آن‌ها را در ادامه توضیح می‌دهیم.

۱.۵.۳ جمعیت اولیه

برای تولید یک جمعیت اولیه از جواب‌ها، np رأس را به تصادف انتخاب می‌کنیم. فرض کنید v_i رأس اول و T درختی باشد که قصد داریم آن را بسازیم. مشابه عملگر ادغام در دو الگوریتم $GKLC$۱ و $GKLC$۲ با $T = v_i$ شروع کرده و مقدار تابع شایستگی آن یعنی $f(T)$ را محاسبه می‌کنیم. سپس رأس v_j را از بین همسایه‌های v_i به تصادف انتخاب می‌کنیم. فرض کنید P مسیری از v_i به v_j باشد. اگر $f(P) < f(T)$ باشد آن‌گاه v_j به درخت T اضافه خواهد شد. این روند ادامه می‌یابد تا زمانی که درخت T به عنوان اولین عضو جمعیت اولیه ساخته شود. اعضای دیگر جمعیت اولیه به روش مشابه تولید می‌شوند. ما در این روش وزن‌های رئوس را درنظر نگرفتیم و آن‌ها را به طور تصادفی انتخاب کردیم.

۲.۵.۳ ادغام

بعد از تولید جمعیت اولیه دو عضو از جمعیت را مشابه با الگوریتم‌های $GKLC$۱ و $GKLC$۲ به عنوان والدین در عملگر ادغام انتخاب می‌کنیم. سپس یکی از یال‌های مشترک والدین مانند $v_i - v_j$ را در فرزندان قرار می‌دهیم. فرض کنید T_{rv_i} و T_{rv_j} دو زیردرخت از والد r باشند که با حذف r به دست می‌آیند. به طوری که $v_i \in T_{rv_i}$ و $v_j \in T_{rv_j}$. اکنون برای ساختن یک درخت در هر یک از فرزندان، $T_{۱v_i}$ را از والد۱ و $T_{۲v_j}$ را از والد۲ در فرزند۱ و $T_{۲v_i}$ از والد۲ و $T_{۱v_j}$ را از والد۱ در فرزند۲ قرار می‌دهیم. ممکن است مسیرهایی با طول بیش از l در فرزندان وجود داشته باشد یا تعداد برگ‌های آن‌ها بیشتر از k باشد. بنابراین زیردرخت‌هایی را که طول بیشتر از l دارند هرس می‌کنیم و برگ‌های اضافی را نیز از فرزندان حذف می‌کنیم. بقیه‌ی الگوریتم مشابه با الگوریتم‌های $GKLC$۱ و $GKLC$۲ می‌باشد.شکل ۱۸.۳ عملیات ادغام را در الگوریتم $GKLC$۳ نشان می‌دهد.



شکل ۱۸.۳: عملیات ادغام در الگوریتم $GKLC$۳.

۶.۳ نتایج محاسباتی

ما الگوریتم‌هایمان را روی ۴۰ نمونه مسأله از $OR - library$ آزمایش می‌کنیم. این نمونه‌ها از مسائل $p-$میانه هستند که در [۶] آمده‌اند. الگوریتم‌ها را به زبان برنامه‌نویسی متلب نوشتیم و ۲۰ بار برای هر مسأله اجرا کردیم. تمام برنامه‌ها روی سیستم با پنتیوم IV و حافظه‌ی ۲ گیگابایت و پردازشگر ۲/۴۰ گیگاهرتز اجرا شده است.

در حالت $k = ۲$ و $l = \infty$ مسأله‌ی $(k, l)-$هسته همان مسأله‌ی هسته است. از طرف دیگر از آن جا که در همه‌ی ۴۰ نمونه یک مسیر همیلتونی وجود دارد و چون

GKLC۳			GKLC۲			GKLC۱					
Time	L.i.	f	Time	L.i.	f	Time	L.i.	f	Edge	n	Test#
۲۰.۷۶۸	۳۶	۶۳۹۶	۱۷.۳۹۰	۱۳	۴۵۱۴	۲۶.۲۳۸	۴۵	۴۸۷۳	۲۰۰	۱۰۰	pmed۱
۱۷.۹۹۱	۳۲	۵۸۴۵	۲۲.۳۶۴	۳۹	۵۸۵۰	۲۷.۵۷۷	۴۵	۶۲۳۸	۲۰۰	۱۰۰	pmed۲
۱۵.۸۹۱	۲۷	۵۸۶۵	۱۵.۶۲۸	۲۶	۵۰۸۴	۱۵.۵۵۳	۲۶	۴۹۶۱	۲۰۰	۱۰۰	pmed۳
۱۵.۷۵۳	۲۶	۸۷۷۳	۲۳.۶۴۶	۴۱	۵۸۹۱	۳۹.۹۰۹	۷۲	۵۶۶۶	۲۰۰	۱۰۰	pmed۴
۲۷.۹۸۰	۴۷	۵۹۸۳	۱۶.۱۷۷	۲۵	۴۴۳۶	۱۹.۰۸۸	۳۰	۴۲۸۶	۲۰۰	۱۰۰	pmed۵
۱۲۵.۹۴۶	۲۹	۶۰۲۴	۲۳۵.۷۶۵	۸۳	۴۱۹۴	۲۷۷.۲۲۱	۹۴	۴۶۲۰	۸۰۰	۲۰۰	pmed۶
۲۳۱.۷۹۲	۵۱	۵۳۳۵	۱۹۹.۲۲۱	۷۰	۴۹۱۱	۷۲.۱۶۳	۲۵	۵۲۳۹	۸۰۰	۲۰۰	pmed۷
۱۲۲.۹۸۵	۲۸	۵۵۵۶	۹۱.۵۵۲	۳۲	۵۵۷۷	۱۲۰.۳۰۱	۴۱	۴۹۱۶	۸۰۰	۲۰۰	pmed۸
۲۵۶.۰۶۴	۵۸	۵۰۰۶	۹۷.۶۷۳	۳۴	۵۰۵۸	۱۴۹.۸۳۱	۵۱	۴۹۶۹	۸۰۰	۲۰۰	pmed۹
۱۵۸.۳۸۴	۳۶	۴۴۳۲	۸۹.۷۹۲	۳۲	۴۵۶۷	۱۹۰.۶۴۴	۶۶	۴۲۱۲	۸۰۰	۲۰۰	pmed۱۰
۶۷۷.۴۵۰	۴۲	۵۲۵۵	۱۲۳۵.۳۹۹	۷۷	۴۸۵۰	۷۷۴.۴۶۶	۴۹	۵۲۱۴	۱۸۰۰	۳۰۰	pmed۱۱
۴۱۶.۸۳۴	۲۸	۵۵۱۸	۷۷۷.۸۲۷	۴۸	۵۵۰۱	۴۹۶.۷۹۴	۳۱	۵۷۵۳	۱۸۰۰	۳۰۰	pmed۱۲
۳۲۲.۸۸۸	۲۹	۵۷۳۹	۱۰۹۶.۴۲۱	۷۵	۵۳۵۴	۱۲۰۳.۲۴۴	۸۵	۴۸۴۶	۱۸۰۰	۳۰۰	pmed۱۳
۴۶۸.۵۶۲	۴۷	۵۸۹۵	۴۷۴.۱۴۸	۲۹	۵۲۸۲	۶۶۸.۷۸۴	۴۰	۴۶۶۰	۱۸۰۰	۳۰۰	pmed۱۴
۵۵۰.۳۱۲	۵۴	۵۶۷۱	۱۲۸۷.۳۰۶	۶۹	۴۹۲۲	۷۴۴.۱۲۶	۴۸	۵۰۲۶	۱۸۰۰	۳۰۰	pmed۱۵
۸۷۴.۹۰۱	۳۵	۵۱۷۲	۱۶۹۳.۸۵۹	۶۴	۵۲۰۴	۱۷۴۸.۴۹۰	۴۳	۵۰۶۱	۳۲۰۰	۴۰۰	pmed۱۶
۵۳۸.۶۵۸	۲۲	۶۲۶۸	۶۴۷.۹۴۴	۲۷	۵۹۵۵	۲۱۸۱.۴۲۵	۸۲	۵۰۶۷	۳۲۰۰	۴۰۰	pmed۱۷
۸۳۷.۷۴۲	۳۴	۵۶۰۲	۱۰۴۵.۶۸۱	۴۳	۵۶۶۵	۱۱۸۳.۲۳۲	۴۴	۵۶۶۳	۳۲۰۰	۴۰۰	pmed۱۸
۶۱۷.۴۵۸	۲۵	۵۵۵۵	۲۲۹۰.۸۳۶	۹۲	۴۹۵۲	۷۹۸.۹۵۴	۳۱	۵۱۲۶	۳۲۰۰	۴۰۰	pmed۱۹
۹۱۸.۷۹۱	۳۵	۵۷۰۶	۶۸۱.۵۲۸	۲۸	۵۸۸۳	۹۴۲.۹۷۱	۳۹	۵۴۶۹	۳۲۰۰	۴۰۰	pmed۲۰
۱۴۹۹.۵۰۲	۳۰	۵۵۹۲	۳۰۹۹.۳۸۶	۵۴	۵۰۳۳	۱۸۵۶.۱۶۱	۳۸	۵۲۷۹	۵۰۰۰	۵۰۰	pmed۲۱
۱۹۴۳.۸۶۴	۳۹	۷۰۱۹	۴۸۱۲.۴۹۳	۶۳	۵۸۱۹	۳۲۶۰.۱۶۹	۴۳	۵۸۵۲	۵۰۰۰	۵۰۰	pmed۲۲
۱۸۴۹.۸۸۸	۳۷	۶۱۶۸	۸۴۶۳.۳۷۵	۶۶	۵۸۱۷	۴۳۰۰.۷۱۲	۵۲	۵۸۳۴	۵۰۰۰	۵۰۰	pmed۲۳
۳۸۴۵.۴۷۴	۵۴	۵۸۳۰	۲۸۲۹.۷۶۰	۵۲	۵۲۶۹	۳۲۲۴.۲۳۳	۶۰	۵۲۱۳	۵۰۰۰	۵۰۰	pmed۲۴
۳۳۳۶.۸۹۹	۴۴	۶۵۸۰	۳۲۲۳.۱۰۸	۴۳	۵۵۷۴	۲۳۳۰.۴۲۳	۳۳	۵۹۰۹	۵۰۰۰	۵۰۰	pmed۲۵
۳۶۶۳.۹۲۲	۴۰	۶۱۱۱	۴۵۲۳.۶۳۳	۴۴	۵۴۴۴	۷۲۲۹.۲۰۱	۵۶	۵۴۴۴	۷۲۰۰	۶۰۰	pmed۲۶
۶۲۷۱.۴۱۳	۴۷	۶۳۳۸	۳۳۲۷.۲۷۶	۳۱	۶۶۹۳	۷۱۹۶.۳۰۰	۷۷	۵۳۱۰	۷۲۰۰	۶۰۰	pmed۲۷
۲۸۱۰.۹۳۰	۳۰	۵۵۶۷	۵۶۴۹.۰۴۲	۴۱	۵۲۸۱	۹۵۷۶.۳۵۲	۷۴	۵۰۸۱	۷۲۰۰	۶۰۰	pmed۲۸
۳۰۲۷.۵۸۲	۳۲	۶۱۸۹	۳۲۱۹.۶۸۱	۳۵	۵۵۱۳	۳۵۹۱.۰۴۶	۳۹	۵۴۴۲	۷۲۰۰	۶۰۰	pmed۲۹
۴۸۴۴.۴۴۶	۵۴	۶۰۴۱	۵۷۳۳.۵۳۴	۶۰	۵۴۸۰	۴۱۵۴.۱۲۶	۴۶	۵۹۰۸	۷۲۰۰	۶۰۰	pmed۳۰
۵۹۴۷.۶۹۲	۲۷	۵۵۱۸	۸۸۷۰.۱۸۰	۵۳	۵۱۷۰	۶۰۶۶.۰۹۸	۳۸	۵۰۹۱	۹۸۰۰	۷۰۰	pmed۳۱
۷۳۳۳.۱۹۴	۳۴	۶۷۶۲	۱۰۳۴۹.۰۹۴	۷۰	۵۸۱۹	۱۰۷۵۵.۶۸۸	۷۳	۵۹۹۵	۹۸۰۰	۷۰۰	pmed۳۲
۳۴۱۱.۷۹۶	۲۱	۶۰۹۹	۶۸۲۰.۳۴۴	۳۴	۵۱۲۶	۱۳۲۲۵.۳۹۰	۳۳	۵۰۰۰	۹۸۰۰	۷۰۰	pmed۳۳
۵۱۴۷.۲۸۳	۲۶	۵۴۴۲	۱۳۷۵۰.۶۵۹	۶۹	۵۵۶۱	۱۲۰۹۸.۵۲۷	۵۵	۵۷۰۱	۹۸۰۰	۷۰۰	pmed۳۴
۷۰۲۷.۳۷۰	۲۵	۶۰۵۵	۸۹۲۳.۴۰۶	۳۴	۶۵۱۳	۱۲۸۴۱.۲۶۷	۶۱	۷۲۶۷	۱۲۸۰۰	۸۰۰	pmed۳۵
۷۴۳۰.۶۰۸	۲۴	۷۰۷۳	۱۱۲۵۲.۵۱۵	۴۳	۷۴۱۵	۲۲۶۹۴.۵۶۱	۸۷	۶۸۱۷	۱۲۸۰۰	۸۰۰	pmed۳۶
۹۸۷۲.۶۳۹	۳۲	۶۸۴۳	۱۰۳۲۴.۱۰۷	۴۱	۶۴۹۱	۲۹۵۱۹.۲۴۸	۱۰۰	۷۲۱۱	۱۲۸۰۰	۸۰۰	pmed۳۷
۱۹۰۲۱.۷۹۲	۴۳	۴۴۸۷	۱۰۳۵۱.۲۵۴	۲۷	۶۴۲۵	۱۶۳۱۴.۸۶۱	۵۰	۵۹۰۸	۱۶۲۰۰	۹۰۰	pmed۳۸
۸۸۷۲.۲۲۰	۲۱	۵۹۳۷	۱۳۶۱۲.۳۵۱	۳۳	۶۳۵۲	۱۶۸۸۶.۵۲۸	۵۰	۶۵۱۶	۱۶۲۰۰	۹۰۰	pmed۳۹
۱۰۱۵۲.۷۱۹	۴۱	۶۹۳۳	۶۵۹۶.۲۷۸	۲۴	۶۸۴۵	۱۱۱۲۸.۴۸۱	۵۰	۶۶۰۷	۱۶۲۰۰	۹۰۰	pmed۴۰

جدول ۴.۳: نتایج به دست آمده از نمونه مسائل با وزن‌های مثبت

GKLC۳			GKLC۲			GKLC۱					
Time	L.i.	f	Time	L.i.	f	Time	L.i.	f	Edge	n	Test#
۱۴.۱۱۹	۲۴	۵۳۳۶	۲۹.۱۰۰	۵۰	۵۰۲۱	۱۹.۸۱۰	۳۴	۵۱۷۶	۲۰۰	۱۰۰	pmed۱
۱۴.۷۷۲	۲۵	۶۰۲۷	۲۲.۲۶۱	۳۹	۴۸۷۴	۲۵.۷۲۶	۴۴	۴۷۷۶	۲۰۰	۱۰۰	pmed۲
۱۶.۱۷۶	۴۳	۴۹۹۸	۱۳.۸۳۴	۲۲	۴۵۷۰	۲۶.۹۷۴	۴۷	۵۲۵۸	۲۰۰	۱۰۰	pmed۳
۱۵.۰۱۱	۴۰	۶۸۸۶	۱۸.۱۸۵	۳۲	۶۵۰۹	۱۴.۱۵۰	۲۴	۶۹۷۱	۲۰۰	۱۰۰	pmed۴
۸.۱۱۰	۲۱	۵۵۲۹	۱۱.۸۷۸	۲۰	۸۲۶۴	۱۱.۶۲۸	۲۰	۸۲۶۴	۲۰۰	۱۰۰	pmed۵
۲۰۳.۱۶۵	۶۶	۵۵۷۶	۱۹۷.۸۰۱	۶۶	۳۸۹۳	۸۵.۲۶۹	۲۸	۳۶۷۳	۸۰۰	۲۰۰	pmed۶
۷۹.۷۵۱	۲۷	۵۲۳۱	۲۵۲.۲۳۹	۸۲	۳۹۸۰	۱۴۷.۶۸۴	۵۲	۴۰۰۹	۸۰۰	۲۰۰	pmed۷
۲۸۳.۹۵۲	۶۱	۵۱۱۱	۱۴۰.۸۳۳	۴۷	۳۹۶۳	۱۸۰.۹۱۴	۶۴	۴۲۰۳	۸۰۰	۲۰۰	pmed۸
۱۲۸.۶۳۶	۲۸	۵۳۳۴	۱۶۶.۰۵۶	۵۳	۳۹۲۴	۲۹۸.۴۱۰	۱۰۰	۳۳۰۲	۸۰۰	۲۰۰	pmed۹
۲۶۳.۱۱۱	۵۷	۴۱۹۲	۱۴۳.۹۹۰	۴۶	۳۵۸۸	۱۸۳.۶۲۷	۶۳	۳۳۳۶	۸۰۰	۲۰۰	pmed۱۰
۴۴۶.۰۰۵	۴۱	۵۰۸۵	۸۴۵.۵۳۴	۵۶	۴۱۹۶	۴۲۵.۶۷۶	۲۷	۴۳۹۰	۱۸۰۰	۳۰۰	pmed۱۱
۵۵۶.۲۹۴	۴۲	۵۰۵۳	۱۱۹۹.۵۲۰	۷۸	۳۹۹۱	۱۴۳۰.۳۰۸	۸۸	۳۶۵۸	۱۸۰۰	۳۰۰	pmed۱۲
۸۱۰.۲۵۲	۴۶	۵۰۶۹	۱۱۰۷.۲۳۳	۷۴	۳۸۶۰	۷۹۹.۰۹۸	۵۰	۳۸۴۵	۱۸۰۰	۳۰۰	pmed۱۳
۷۹۸.۶۱۰	۴۸	۵۲۸۸	۸۲۲.۶۷۷	۶۲	۴۱۱۵	۸۱۵.۲۹۰	۵۳	۴۱۵۵	۱۸۰۰	۳۰۰	pmed۱۴
۷۹۹.۸۱۵	۴۸	۵۳۹۹	۱۰۹۵.۷۷۰	۱۰۰	۴۸۱۳	۱۴۸۳.۸۵۹	۸۷	۴۴۹۵	۱۸۰۰	۳۰۰	pmed۱۵
۸۵۶.۴۱۷	۳۴	۵۵۷۵	۲۹۶۵.۶۵۷	۷۷	۴۱۳۶	۳۳۴۰.۸۴۵	۸۰	۳۸۴۴	۳۲۰۰	۴۰۰	pmed۱۶
۱۲۷۱.۸۷۶	۵۱	۵۲۹۲	۲۱۰۵.۳۱۶	۵۸	۳۹۲۴	۷۷۲.۸۴۷	۲۹	۳۹۵۵	۳۲۰۰	۴۰۰	pmed۱۷
۷۸۲.۸۳۱	۳۱	۵۰۱۳	۷۱۱.۶۹۱	۲۷	۴۶۴۸	۱۹۶۱.۵۷۰	۶۹	۴۳۶۹	۳۲۰۰	۴۰۰	pmed۱۸
۵۷۴.۳۱۵	۲۱	۵۰۲۰	۷۹۹.۴۰۵	۳۱	۴۰۸۵	۱۳۸۹.۴۹۲	۳۵	۴۱۲۱	۳۲۰۰	۴۰۰	pmed۱۹
۱۳۵۶.۶۷۳	۵۲	۵۱۹۲	۱۲۴۶.۹۹۰	۴۸	۴۴۷۱	۴۰۲۱.۰۴۱	۱۰۰	۴۰۶۹	۳۲۰۰	۴۰۰	pmed۲۰
۳۱۳۰.۵۰۶	۵۳	۴۸۳۴	۲۷۰۱.۷۸۹	۴۵	۴۰۲۵	۱۷۴۵.۰۸۶	۳۴	۴۵۳۴	۵۰۰۰	۵۰۰	pmed۲۱
۲۵۷۰.۹۵۲	۲۹	۶۸۳۸	۷۹۰۳.۷۰۷	۱۰۰	۴۴۲۸	۷۳۶۵.۵۳۸	۹۹	۴۶۹۲	۵۰۰۰	۵۰۰	pmed۲۲
۱۷۳۳.۳۱۴	۲۱	۵۵۴۷	۲۰۸۱.۶۶۵	۲۶	۴۴۰۳	۵۹۱۹.۸۸۶	۹۲	۴۲۵۳	۵۰۰۰	۵۰۰	pmed۲۳
۳۲۳۸.۳۶۹	۳۵	۵۷۴۶	۴۱۷۹.۱۵۷	۷۸	۴۲۵۹	۲۵۳۲.۲۸۷	۴۸	۴۳۴۲	۵۰۰۰	۵۰۰	pmed۲۴
۳۶۷۱.۶۵۵	۴۲	۷۵۲۶	۲۰۱۵.۵۶۷	۳۶	۴۶۲۸	۱۷۳۴.۸۷۲	۳۱	۴۷۹۵	۵۰۰۰	۵۰۰	pmed۲۵
۵۷۸۶.۴۶۷	۴۵	۵۷۲۱	۵۱۱۶.۳۸۲	۵۵	۴۴۷۸	۶۰۷۶.۶۴۴	۵۳	۴۴۶۴	۷۲۰۰	۶۰۰	pmed۲۶
۵۴۸۶.۱۴۸	۴۳	۵۰۴۶	۷۴۱۱.۴۴۶۳	۵۶	۴۰۷۶	۵۷۰۶.۷۵۲	۵۴	۳۹۸۲	۷۲۰۰	۶۰۰	pmed۲۷
۴۶۷۵.۵۷۰	۳۷	۴۹۸۹	۱۲۲۵۶.۲۴۹	۱۰۰	۴۱۲۸	۱۲۴۷۷.۴۶۵	۸۹	۳۹۷۲	۷۲۰۰	۶۰۰	pmed۲۸
۳۹۶۰.۴۵۵	۳۸	۵۱۷۹	۷۱۳۸.۰۴۱	۵۹	۴۶۵۰	۱۹۰۵۷.۷۹۷	۸۵	۴۱۱۸	۷۲۰۰	۶۰۰	pmed۲۹
۳۸۴۱.۶۹۹	۴۳	۷۴۷۴	۸۲۷۰.۵۸۹	۹۴	۴۷۷۱	۲۲۲۴.۸۷۵	۲۳	۶۰۴۳	۷۲۰۰	۶۰۰	pmed۳۰
۴۳۲۹.۱۵۷	۳۰	۵۰۱۲	۱۱۷۵۷.۵۴۷	۶۹	۴۲۶۳	۸۹۵۲.۶۹۴	۶۰	۴۱۹۴	۹۸۰۰	۷۰۰	pmed۳۱
۵۱۲۳.۴۷۶	۳۶	۵۰۴۷	۱۱۷۶۸.۱۸۸	۷۸	۴۴۷۷	۱۶۱۱۳.۳۹۱	۷۷	۴۶۰۷	۹۸۰۰	۷۰۰	pmed۳۲
۵۶۴۳.۵۲۰	۲۵	۵۲۴۲	۱۱۷۷۲.۲۲۴	۷۹	۴۸۳۹	۱۷۰۴۵.۶۱۹	۸۲	۴۳۲۰	۹۸۰۰	۷۰۰	pmed۳۳
۹۳۹۴.۵۱۸	۴۵	۶۲۰۷	۱۷۶۹۹.۰۵۵	۶۲	۴۵۷۲	۱۰۶۸۷.۰۱۲	۵۷	۴۴۵۱	۹۸۰۰	۷۰۰	pmed۳۴
۸۹۷۴.۷۴۴	۲۷	۴۸۵۲	۵۰۱۲.۳۲۲	۳۴	۴۳۹۶	۵۰۱۰.۳۱۲	۲۹	۴۳۶۹	۱۲۸۰۰	۸۰۰	pmed۳۵
۸۷۵۴.۰۴۴	۲۶	۵۸۶۸	۱۳۰۹۰.۲۵۰	۳۸	۴۹۸۴	۸۹۵۶.۲۱۱	۴۰	۵۰۲۷	۱۲۸۰۰	۸۰۰	pmed۳۶
۲۰۰۱۵.۰۲۷	۳۴	۵۷۵۱	۱۸۲۰۴.۳۲۲	۵۶	۴۹۸۷	۱۶۵۷۵.۱۴۵	۴۳	۵۰۷۰	۱۲۸۰۰	۸۰۰	pmed۳۷
۱۰۹۲۰.۱۳۹	۲۳	۵۴۹۷	۱۱۲۲۵.۹۴۶	۳۳	۴۵۶۸	۲۱۳۴۳.۸۳۱	۸۷	۴۴۹۱	۱۶۲۰۰	۹۰۰	pmed۳۸
۶۷۸۲.۹۹۳	۲۱	۵۱۳۰	۳۱۵۵۴.۹۶۹	۶۴	۴۵۷۷	۱۸۶۹۰.۹۱۹	۶۸	۴۳۹۷	۱۶۲۰۰	۹۰۰	pmed۳۹
۱۲۰۲۱.۲۰۸	۲۵	۶۰۳۹	۱۱۸۱۱.۱۰۵	۲۵	۵۲۹۰	۱۴۷۴۲.۵۳۱	۴۰	۵۱۷۸	۱۶۲۰۰	۹۰۰	pmed۴۰

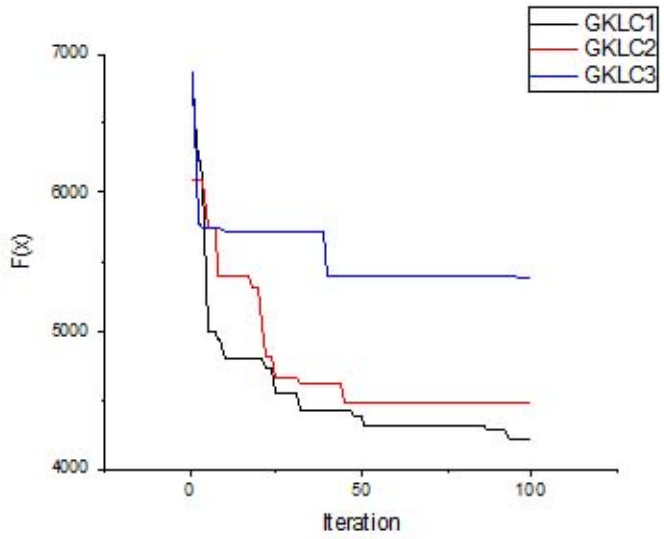
جدول ۵.۳: نتایج به دست آمده از نمونه مسائل نیمه‌ناخوشایند

مسیر همیلتونی مسیری است که از همه‌ی رئوس می‌گذرد، مقدار تابع هدف برای مسأله‌ی هسته صفر است. بنابراین برای آزمایش کارایی الگوریتم‌هایمان، آن‌ها را در این حالت بررسی کردیم. همه‌ی الگوریتم‌ها جواب بهینه یعنی صفر را به دست آوردند. همچنین الگوریتم‌ها را برای مقادیر قابل تغییر $k = ۰/۱n$ و $l = ۰/۱(۲۰n)$ در حالتی که همه‌ی وزن‌ها مثبت و همچنین در حالت نیمه‌ناخوشایند امتحان کردیم. در حالت وزن مثبت، همه‌ی وزن‌ها را ۱ و در حالت نیمه‌ناخوشایند ده وزن اول را ۱− و بقیه‌ی آن‌ها را ۱ درنظر می‌گیریم. نتایج در جداول ۴.۳ و ۵.۳ ارائه شده‌اند. در این جدول‌ها، ستون f مقدار تابع هدف را نشان می‌دهد. ستون $L.i.$، تعداد تکرارها را بعد از اینکه هیچ بهبودی در جواب بعد از ۲۰ تکرار به دست نیامد بیان می‌کند و ستون $Time$ زمان اجرای الگوریتم‌ها را برای ۱۰ تکرار نشان می‌دهد.
برای توضیح بیشتر فرض کنید مقادیر $L.i. = ۵۰$ و $f = ۶۵۱۶$ را داریم. این مقادیر بیان می‌کنند که جواب ۶۵۱۶ بعد از ۳۰ بار اجرای الگوریتم به دست آمده است و تا ۲۰ بار تکرار بعد از آن تغییری نکرده است. بنابراین تعداد تکرار الگوریتم ۵۰ است. ستون‌های آخر این جداول زمان اجرای الگوریتم را به ثانیه نشان می‌دهد.

۱.۶.۳ مقایسه‌ی الگوریتم‌ها

در این قسمت مقدار بهینه‌ی تابع هدف را در سه الگوریتم مقایسه می‌کنیم. از آن جا که در $GKLC۱$ و $GKLC۲$ رئوس با بالاترین وزن را انتخاب می‌کنیم، بنابراین مقدار بهینه‌ی تابع هدف بهتر از $GKLC۳$ است. همچنین چون در $GKLC۳$ رئوس را به تصادف انتخاب می‌کنیم بنابراین سرعت اجرای الگوریتم در $GKLC۳$ بیشتر از $GKLC۱$ و $GKLC۲$ است. از طرف دیگر عملگر ادغام در $GKLC۱$ و $GKLC۲$ همه‌ی یال‌های مشترک والدین را در فرزندان قرار می‌دهد اما در $GKLC۳$ تنها یکی از یال‌های مشترک والدین در بچه‌ها قرار داده می‌شود. بنابراین جواب بهینه در $GKLC۱$ و $GKLC۲$ از $GKLC۳$ بهتر است. همچنین در $GKLC۱$ و $GKLC۲$ زمان زیادی صرف اتصال رئوس غیرمتصل می‌شود. در نتیجه سرعت اجرای $GKLC۳$ بیشتر از $GKLC۱$ و $GKLC۲$ است. از ۸۰ مسأله‌ی آزمایش‌شده، در ۴۳ مسأله، بهترین جواب با الگوریتم $GKLC۱$، ۳۳ مسأله با الگوریتم $GKLC۲$ و برای ۷ مسأله توسط الگوریتم $GKLC۳$ به دست آمده است.
شکل ۱۹.۳ نمودار بهبود جواب را در صد تکرار الگوریتم برای نمونه مسأله‌ی $pmed۱۵$ در جدول ۵.۳ در هر سه الگوریتم $GKLC۱$، $GKLC۲$ و $GKLC۳$ نشان می‌دهد.

در بخش بعد برای تجزیه و تحلیل تأثیر پارامترها بر همگرایی الگوریتم‌ها، الگوریتم‌هایمان را برای اندازه‌های مختلف جمعیت اولیه و مقادیر مختلف شرط توقف، مورد سنجش قرار می‌دهیم.



شکل ۱۹.۳: مقدار تابع هدف برای نمونه pmed۱۵ در ۱۰۰ تکرار الگوریتم‌های $GKLC$۱، $GKLC$۲ و $GKLC$۳.

۲.۶.۳ اندازه‌ی جمعیت اولیه و شرط توقف

انتخاب یک اندازه‌ی مناسب برای جمعیت اولیه به طور مستقیم روی کیفیت جواب مسأله تأثیر می‌گذارد. ما الگوریتم‌ها را برای جمعیت اولیه با اندازه‌های مختلف $NP = ۵٪ n$، $NP = ۱۰٪ n$، $NP = ۱۵٪ n$، $NP = ۲۰٪ n$ و $NP = ۲۵٪ n$ روی نمونه مسائل ۱ تا ۲۰ بررسی کردیم. در جدول ۶.۳ نتایج برای $GKLC$۱، ۲۰ بار بعد از آخرین بهبود ارائه شده است. این نتایج نشان می‌دهد که اندازه‌ی جمعیت اولیه‌ی کوچک روی کیفیت جواب‌ها و جمعیت اولیه‌ی بزرگ روی سرعت الگوریتم تأثیر دارد. بر طبق این نتایج اندازه‌ی مناسب جمعیت اولیه برای مسأله‌ی ما $NP = ۱۵٪ n$ است. بنابراین برای همه‌ی اجراهایی که گزارش آن‌ها در جداول ۴.۳ و ۵.۳ آمده است $NP = ۱۵٪ n$ را درنظر می‌گیریم.

همچنین مقدار بهینه‌ی توابع هدف را برای ۲۰ نمونه مسأله در تکرارهای مختلف بررسی کردیم. جدول ۷.۳ نتایج را برای تکرارهای ۵، ۱۰، ۱۵، ۲۰، ۵۰ و ۱۰۰ نشان می‌دهد. در این جدول $iter$ به معنای تکرار است.

۳.۶.۳ مقایسه‌ی سرعت الگوریتم‌ها

از آن جا که در $GKLC$۳ رئوس به تصادف انتخاب می‌شوند، بنابراین سرعت اجرای $GKLC$۳ از $GKLC$۱ و $GKLC$۲ بیشتر است. ما هر سه الگوریتم را برای ۲۰ تکرار روی نمونه مسائل مختلف اجرا کردیم. جدول ۸.۳ این نتایج را نشان می‌دهد. در این

$NP=۲۵٪n$	$NP=۲۰٪n$	$NP=۱۵٪n$	$NP=۱۰٪n$	$NP=۵٪n$	Test#
۴۹۳۶	۵۸۵۵	۴۸۷۳	۵۶۶۰	۶۸۰۳	pmed۱
۶۷۰۵	۶۵۰۳	۶۲۳۸	۶۶۲۰	۷۱۷۶	pmed۲
۵۱۸۵	۴۹۶۱	۴۹۶۱	۵۱۰۳	۵۲۷۷	pmed۳
۶۰۸۰	۵۹۱۸	۵۶۶۶	۶۴۲۲	۷۳۹۸	pmed۴
۴۵۳۱	۴۴۵۹	۴۲۸۶	۵۴۸۲	۶۰۱۳	pmed۵
۴۷۹۰	۴۷۷۴	۴۶۲۰	۵۰۹۹	۴۹۲۰	pmed۶
۵۵۰۴	۵۶۳۰	۵۲۳۹	۵۳۷۲	۵۶۰۷	pmed۷
۵۲۵۶	۵۱۹۷	۴۹۱۶	۵۴۷۲	۵۳۸۴	pmed۸
۴۹۸۳	۴۸۸۴	۴۶۶۹	۴۹۱۰	۵۲۲۸	pmed۹
۴۴۵۶	۴۴۵۶	۴۲۱۲	۴۵۸۰	۴۹۲۱	pmed۱۰
۵۸۰۱	۵۳۴۴	۵۲۱۴	۵۴۷۶	۵۳۵۹	pmed۱۱
۵۷۹۱	۵۷۵۳	۵۷۵۳	۵۷۷۲	۵۹۹۳	pmed۱۲
۵۴۶۹	۵۴۱۸	۴۸۴۶	۵۴۹۰	۵۳۴۵	pmed۱۳
۴۹۹۱	۵۴۶۶	۴۶۶۰	۴۹۴۰	۴۹۶۸	pmed۱۴
۵۲۸۹	۵۲۶۲	۵۰۲۶	۵۰۴۱	۵۳۳۲	pmed۱۵
۵۳۹۰	۵۱۰۴	۵۰۶۱	۵۴۰۵	۵۵۰۷	pmed۱۶
۵۱۳۲	۵۲۸۰	۵۰۶۷	۵۴۷۶	۵۵۵۱	pmed۱۷
۵۶۸۵	۵۶۷۳	۵۶۶۳	۵۶۶۳	۵۷۰۵	pmed۱۸
۵۴۸۳	۵۳۳۲	۵۱۲۶	۵۲۱۰	۵۴۸۳	pmed۱۹
۶۲۱۰	۵۸۶۹	۵۴۶۹	۵۶۴۶	۵۸۶۰	pmed۲۰

جدول ۶.۳: نتایج به دست آمده از نمونه مسائل با وزن‌های مثبت در $GKLC$۱ برای بررسی اندازه‌ی جمعیت اولیه.

iter=۱۰۰	iter=۵۰	iter=۲۰	iter=۱۰	iter=۵	Test#
۴۷۲۵	۴۷۲۵	۵۴۵۹	۶۰۹۵	۶۰۹۵	pmed۱
۶۲۶۲	۶۲۶۲	۶۹۸۵	۹۴۷۴	۹۴۷۴	pmed۲
۴۹۰۴	۴۹۰۴	۵۱۸۷	۵۱۸۹	۵۲۶۱	pmed۳
۵۹۲۴	۵۹۲۴	۶۴۲۵	۶۹۳۸	۸۲۶۵	pmed۴
۴۹۷۲	۴۹۷۲	۴۹۷۲	۵۴۸۶	۷۱۲۵	pmed۵
۴۱۵۶	۴۶۸۳	۵۰۹۴	۵۹۷۴	۷۴۸۵	pmed۶
۴۶۲۹	۴۶۹۸	۵۲۶۷	۵۳۵۹	۶۵۷۲	pmed۷
۵۰۶۹	۵۱۷۶	۵۲۴۵	۵۴۶۸	۶۹۴۰	pmed۸
۴۹۴۱	۵۰۰۲	۵۲۲۱	۵۲۲۱	۶۸۳۰	pmed۹
۳۶۹۲	۳۷۹۸	۴۳۴۷	۵۵۶۸	۵۵۶۸	pmed۱۰
۴۶۳۲	۵۴۲۳	۵۶۸۹	۷۳۵۶	۷۳۶۵	pmed۱۱
۴۹۹۹	۵۱۱۴	۵۶۹۱	۵۶۹۱	۷۴۲۲	pmed۱۲
۴۷۲۸	۴۷۸۲	۵۱۱۱	۷۰۷۴	۷۰۷۴	pmed۱۳
۴۸۹۷	۴۹۸۹	۶۱۱۰	۶۱۱۰	۷۰۹۵	pmed۱۴
۵۰۳۰	۵۲۵۱	۵۲۵۱	۶۸۱۹	۶۹۰۷	pmed۱۵
۵۱۶۰	۵۱۷۳	۵۴۳۹	۵۴۳۹	۵۶۱۸	pmed۱۶
۴۸۸۳	۵۴۸۴	۵۷۳۱	۵۷۳۱	۷۲۶۴	pmed۱۷
۵۶۱۶	۵۶۱۶	۵۷۹۵	۵۷۹۵	۷۷۴۶	pmed۱۸
۴۵۹۷	۴۸۱۳	۵۱۱۸	۷۵۹۴	۷۷۳۰	pmed۱۹
۵۷۳۴	۵۷۳۴	۵۷۸۴	۸۰۸۴	۸۴۹۵	pmed۲۰

جدول ۷.۳: نتایج به دست آمده از نمونه مسائل با وزن‌های مثبت در $GKLC$۱ برای بررسی مقدار بهینه‌ی تابع هدف با مقادیر مختلف شرط توقف.

Semi − obnoxious problems			*problems with positive weights*			
GKLC۳	GKLC۲	GKLC۱	GKLC۳	GKLC۲	GKLC۱	Test#
۷.۷۲۸	۷.۸۹۷	۷.۸۲۷	۷.۹۰۱	۷.۹۹۱	۸.۱۳۶	pmed۱
۷.۶۴۳	۷.۸۶۲	۷.۸۲۷	۷.۷۲۱	۷.۸۴۴	۷.۸۹۱	pmed۲
۷.۶۸۲	۷.۷۶۸	۷.۸۱۶	۷.۸۲۰	۷.۹۵۷	۷.۹۵۷	pmed۳
۷.۷۱۹	۷.۸۲۱	۷.۸۴۲	۷.۷۷۱	۷.۹۲۱	۸.۰۲۲	pmed۴
۷.۷۴۲	۷.۷۶۱	۷.۷۱۰	۷.۶۷۵	۷.۹۸۸	۷.۹۸۲	pmed۵
۵۶.۹۰۱	۵۷.۳۸۲	۵۸.۳۷۵	۵۸.۵۰۳	۵۸.۶۲۳	۵۹.۰۱۱	pmed۶
۵۶.۸۲۰	۵۸.۰۱۴	۵۷.۴۳۰	۵۷.۹۷۹	۵۸.۴۰۷	۵۸.۷۳۲	pmed۷
۵۷.۱۷۴	۵۸.۵۸۵	۵۷.۸۱۹	۵۷.۹۴۶	۵۸.۴۰۹	۵۸.۳۱۸	pmed۸
۵۷.۱۱۴	۵۷.۶۱۶	۵۸.۲۱۲	۵۷.۸۲۳	۵۸.۴۱۰	۵۸.۰۱۴	pmed۹
۵۷.۱۳۰	۵۸.۲۷۱	۵۸.۰۶۶	۵۷.۵۸۱	۵۸.۴۷۳	۵۸.۵۶۷	pmed۱۰
۱۹۵.۲۹۸	۲۰۷.۹۹۲	۲۰۸.۲۷۱	۱۹۳.۳۴۱	۱۹۵.۲۴۱	۲۱۱.۰۰۴	pmed۱۱
۱۹۴.۵۴۳	۱۹۵.۰۲۴	۱۹۷.۲۱۳	۱۹۳.۴۹۹	۱۹۴.۷۸۱	۱۹۵.۱۶۹	pmed۱۲
۱۹۶.۱۱۷	۲۰۲.۹۶۵	۲۰۳.۵۸۵	۱۹۴.۷۸۱	۲۰۲.۶۷۶	۲۰۳.۲۰۹	pmed۱۴
۵۰۸.۸۳۰	۵۱۲.۰۲۰	۵۲۰.۰۴۳	۴۸۳.۶۹۰	۴۸۶.۶۰۸	۴۸۸.۴۲۴	pmed۱۶
۵۰۸.۴۷۳	۵۶۴.۲۸۹	۵۱۵.۴۹۵	۴۹۲.۹۲۷	۵۰۰.۳۴۰	۴۹۵.۵۸۶	pmed۱۸
۵۵۱.۲۵۵	۵۵۳.۳۶۴	۵۵۲.۷۴۰	۴۷۹.۷۸۹	۴۹۶.۵۲۳	۵۰۱.۹۰۲	pmed۲۰
۹۶۵.۹۶۴	۱۰۳۶.۲۸۰	۱۲۲۳.۴۷۷	۱۶۴۹.۲۹۵	۱۶۶۹.۱۵۰	۱۷۲۷.۴۱۸	pmed۲۳
۱۸۳۱.۵۷۱	۲۳۷۳.۵۵۴	۳۰۴۶.۳۱۴	۱۷۷۲.۵۴۱	۲۷۰۸.۸۱۰	۲۵۰۱.۹۷۰	pmed۲۶
۲۶۷۹.۰۳۶	۲۹۹۵.۷۳۶	۲۹۰۷.۶۲۳	۱۷۹۹.۵۲۴	۱۹۹۴.۰۵۸	۲۸۴۰.۶۸۰	pmed۳۰
۴۶۸۵.۲۰۴	۵۸۶۱.۷۵۷	۶۱۲۸.۱۰۱	۳۹۵۰.۲۶۹	۴۱۸۷.۶۹۷	۴۰۷۳.۸۳۶	pmed۳۵
۶۱۴۴.۰۸۹	۶۳۵۶.۱۷۰	۷۴۵۹.۰۳۷	۵۸۱۷.۵۵۲	۵۹۶۴.۱۸۶	۵۹۹۱.۶۷۲	pmed۴۰

جدول ۸.۳: مدت زمان اجرای الگوریتم بعد از ۲۰ تکرار برای نمونه مسائل مختلف.

بررسی مسائل با وزن مثبت و همچنین مسائل نیمه ناخوشایند را در نظر می‌گیریم. همان طور که انتظار داشتیم در تمام نمونه مسائل زمان اجرای الگوریتم $GKLC$۳ کمتر از $GKLC$۱ و $GKLC$۲ می‌باشد.

۷.۳ نتیجه‌گیری

در این فصل مسأله‌ی پیداکردن $(k,l)-$هسته‌ی یک شبکه با وزن مثبت و منفی(نیمه ناخوشایند) در نظر گرفته شد. از آن‌جا که این مسأله یک مسأله‌ی $NP-$سخت است، برای حل آن از الگوریتم ژنتیک استفاده کردیم. سه الگوریتم ژنتیک طراحی و نتایج عددی به دست آمده از آن‌ها بررسی شد. همچنین الگوریتم ها از نظر کیفیت جواب و سرعت همگرایی با هم مقایسه شدند. در مطالعات بعدی می‌توان این مسأله را با الگوریتم‌های ابتکاری دیگر از جمله الگوریتم مورچگان و پرندگان حل کرد و نتایج عددی به دست آمده از آن‌ها را با هم و با الگوریتم ژنتیک مقایسه کرد.

فصل ۴

۲ − (k, l) − هسته‌ی یک درخت

۱.۴ مقدمه

در این فصل مسأله‌ی پیداکردن ۲ − (k, l) − هسته‌ی یک درخت مانند T که وزن‌های رئوس در T می‌تواند مثبت یا منفی باشند را مورد بررسی قرار می‌دهیم. یک ۲ − (k, l) − هسته‌ی یک درخت، دو زیردرخت منفصل T_1 و T_2 از T است که هر یک از آن‌ها باید حداکثر k برگ و قطر حداکثر l داشته باشند، به طوری که مجموع فاصله‌ها از همه‌ی رئوس تا این زیردرخت‌ها کمینه باشد. در ادامه ابتدا مدل مسأله را بیان کرده و سپس درخت بدون وزن را در نظرمی‌گیریم و نشان می‌دهیم جوابی وجود دارد که هیچ یک از دو زیردرخت ۲ − (k, l) − هسته یک رأس نیستند. بعد از آن نشان می‌دهیم زمانی که مجموع وزن‌های رئوس منفی هستند، ۲ − (k, l) − هسته‌ی T می‌تواند با حذف‌کردن یالی که متصل به یک برگ است به دست آید. در پایان یک الگوریتم برای پیداکردن ۲ − (k, l) − هسته‌ی یک درخت با وزن مثبت/منفی ارائه می‌دهیم. این الگوریتم تعمیمی از الگوریتم ارائه‌شده توسط بکر در [۸] می‌باشد.

۲.۴ فرمول‌بندی مسأله

فرض کنید $T=(V,E)$ درخت داده شده با $|V|=n$ باشد. همچنین فرض کنید $w(v_i)$ وزن رأس $v_i \in V$ است. برای سادگی w_i را به جای $w(v_i)$ استفاده می‌کنیم. در نتیجه وزن درخت T که با $w(T)$ نشان می‌دهیم، برابر است با $w(T)=\sum_{i=1}^{n} w_i$.
طول یال (i,j) را $a(i,j)$ در نظر می‌گیریم و طول مسیر از v_i به v_j را با $d(v_i,v_j)$ نشان می‌دهیم. بنابراین طول کوتاه‌ترین مسیر بین T_1 و رأس v به صورت زیر به دست می‌آید،

$$d(v,T_1)=min_{u\in T_1}\ d(u,v).$$

طول هر مسیر مانند P را با $L(P)$ و وزن زیردرخت T' از T را با $w(T')$ نشان می‌دهیم که برابر است با $w(T')=\sum_{v_i\in T'} w_i$.
همان طور که قبلاً ذکر شد، یک (k,l)–هسته‌ی T زیردرخت T_1 از آن با حداکثر k برگ و قطر حداکثر l است، به طوری که مجموع فاصله‌ها از همه‌ی رئوس تا T_1 کمینه شود، یعنی

$$minF(T_1)=\sum_{v_i\in V\setminus V_1} w_i d(v_i,T_1)$$

حال به طور مشابه ۲ – (k,l)–هسته‌ی یک درخت را تعریف می‌کنیم. یک – (k,l)–۲هسته‌ی T مجموعه‌ای از دو زیردرخت $T_1=(V_1,E_1)$ و $T_2=(V_2,E_2)$ است که هر یک از آن‌ها حداکثر k برگ دارد و طولشان حداکثر به اندازه‌ی l است، به طوری که تابع زیر کمینه شود،

$$F(T_1,T_2)=\sum_{v_i\in V\setminus(V_1\cup V_2)} w_i d(v_i,T_1,T_2)$$

که در آن $d(v_i,T_1,T_2)$ کمترین فاصله رأس v_i تا T_1 و T_2 است. کاربردهای – (k,l)–۲هسته مشابه (k,l)–هسته می‌باشد. برای کاربرد ۲ – (k,l)–هسته در حالت نیمه ناخوشایند به [۴۸] و [۶۳] مراجعه کنید.

۳.۴ ویژگی‌هایی برای حالات خاص

در این بخش برخی از ویژگی‌های ۲ – (k,l)–هسته را در حالات خاص بررسی می‌کنیم. ایده‌ی اصلی برای حل مسأله‌ی ۲ – (k,l)–هسته روش حذف یال است. در این روش برای یافتن جواب بهینه یک یال را حذف می‌کنیم و سپس (k,l)–هسته‌ی هر یک از دو زیردرختی که از حذف یال به دست می‌آیند را پیدا می‌کنیم. در قضیه‌های زیر برای نشان دادن ویژگی‌های مسأله از این روش استفاده می‌کنیم. در ابتدا درخت بدون

وزن را در نظر می‌گیریم. قضیه‌ی زیر نشان می‌دهد که هیچ کدام از دو زیردرخت ۲ − (k,l)−هسته روی یک درخت بدون وزن با $d_T > ۲$ یک رأس تنها نمی‌باشد.

قضیه ۱.۳.۴. فرض کنید $T = (V, E)$ یک درخت بدون وزن باشد. اگر $l > ۰$ و $d_T > l+۱$ آن گاه یک ۲ − (k,l)−هسته‌ی T شامل دو زیردرخت $T'_۱$ و $T'_۲$ موجود است به قسمی که هیچ یک از $T'_۱$ و $T'_۲$ یک رأس تنها نیستند.

برهان. ابتدا نشان می‌دهیم که یک برگ نمی تواند یکی از دو زیردرخت ۲−(k,l)−هسته باشد. فرض کنید q یک یال بین یک رأس داخلی و یک برگ مانند v_i باشد. با حذف یال q دو زیردرخت $T_۱$ و $T_۲$ به دست می‌آید و اگر $T'_۱ = \{v_i, \emptyset\}$ و $T'_۲ = v_i$ به ترتیب (k,l)−هسته ی $T_۱$ و $T_۲$ باشند و رأس $v_j \in T_۱$ همسایه با v_i باشد، آن گاه اگر $v_j \notin T'_۱$ با اضافه‌کردن v_j به $T'_۲$ مقدار تابع هدف افزایش پیدا نمی‌کند.
در حالتی که $v_j \in T'_۱$ فرض کنید $v_r \in T'_۱$ یک رأس غیربرگ همسایه با v_j باشد و اگر $T'_۱(v_j)$ زیردرختی از $T'_۱$ شامل v_j که از حذف یال (v_j, v_r) به دست می‌آید باشد، آن گاه با اضافه‌کردن همه‌ی رئوس $T'_۱(v_j)$ به $T'_۲$ و حذف آن‌ها از $T'_۱$ مقدار تابع هدف افزایش پیدا نمی‌کند. از آن جا که $d_T > l+۱$، رأس غیر برگ $v_r \in T'_۱$ همسایه با v_j وجود دارد. در حالتی که یکی از (k,l)−هسته یک رأس داخلی باشد به وضوح می‌توان آن را به طرف یک برگ ادامه داد و مقدار تابع هدف را کاهش داد.
بنابراین ۲ − (k,l)−هسته‌ای وجود دارد که هیچ یک از (k,l)−هسته‌اش یک رأس نیستند. □

نکته قابل توجه این که قضیه‌ی ۱.۳.۴ برای حالتی که $d_T \leq ۲$ صدق نمی‌کند. یک مثال نقض برای این حالت گراف استار است. گراف استار درختی است که فقط یک رأس با درجه‌ی بیشتر از یک دارد.
اکنون حالتی را که $w(T) < ۰$ در نظر می‌گیریم. در این حالت طبق قضیه‌ی۲.۳.۲ و لم ۴.۳.۲، (k,l)−هسته یک رأس تنها است که ممکن است یک برگ باشد. اثبات آن‌ها در فصل دوم و در [۴۸] نیز آمده است.

قضیه ۲.۳.۴. اگر T یک درخت با $w(T) < ۰$ باشد، آن گاه جوابی برای مسأله‌ی ۲ − (k,l)−هسته وجود دارد که یکی از (k,l)−هسته هایش یک رأس تنها است.

برهان. فرض کنید $T'_۱$ و $T'_۲$، ۲ − (k,l)−هسته‌ی درخت T باشند. همچنین فرض کنید $T_۱$ و $T_۲$ دو زیردرخت از T باشند که به ترتیب شامل همه‌ی رئوسی که به $T'_۱$ و $T'_۲$ تخصیص داده شده‌اند می‌باشند. بنابراین $T'_۱$ و $T'_۲$ به ترتیب (k,l)−هسته‌ی $T_۱$ و $T_۲$ هستند. از آن جا که $w(T) < ۰$، حداقل یکی از $T_۱$ یا $T_۲$ دارای وزن منفی هستند. بنابراین طبق قضیه‌ی ۲.۳.۲، $T'_۱$ یا $T'_۲$ یا هر دو یک رأس می‌باشند. □

۴.۴ الگوریتم

همان طور که قبلاً ذکر شده است، برای پیداکردن یک ۲ – (k,l)–هسته‌ی درخت T، یالی از آن را حذف و T را به دو قسمت $T_۱$ و $T_۲$ تقسیم می‌کنیم. سپس (k,l)–هسته‌ی هر یک از $T_۱$ و $T_۲$ را پیدا کرده و این زیربرنامه را برای هر یال از T تکرار می‌کنیم. در نهایت بهترین جفت از (k,l)–هسته‌ها را به عنوان ۲ – (k,l)–هسته در نظر می‌گیریم. برای حالت $w(T) < ۰$، طبق لم ۴.۳.۲، (k,l)–هسته یک رأس است. به طوری که این رأس یا یک برگ است و یا یک رأس داخلی مانند u که $w(u) \geq ۰$ و برای هر رأس v همسایه با u، $w(T_{uv}) \leq ۰$ می‌باشد. همچنین در حالت $w(T) > ۰$ می‌توان قضیه‌ی ۴.۳.۲ را به کار برد. بنابراین با استفاده از مطالب فوق الگوریتم زیر را ارائه می‌کنیم:

الگوریتم

ورودی: درخت T با وزن مثبت/منفی

خروجی: یک ۲ – (k,l)–هسته‌ی T مانند $(S_۱^*, S_۲^*)$ و تابع هدف آن d^*

شروع

۱ـ قرار دهید $d^* := +\infty$.

۲ـ قرار دهید $(S_۱^*, S_۲^*) := (\emptyset, \emptyset)$.

۳ـ **برای هر** زیردرخت $T_۱$ و $T_۲$ به دست آمده از T با حذف هر یال $e \in E$ انجام دهید،

۳-۱ـ $SUBTREE(T_۱)$

۳-۲ـ $d_۱ := d(S')$ و $S_۱ := S'$.

۳-۳ـ $SUBTREE(T_۲)$

۳-۴ـ $d_۲ := d(S')$ و $S_۲ := S'$.

۳-۵ـ $d = d_۱ + d_۲$

۳-۶ـ **اگر** $d < d^*$ **آن‌گاه** قرار دهید $d^* := d$ و $(S_۱^*, S_۲^*) := (S_۱, S_۲)$.

پایان

که در آن زیربرنامه‌ی $SUBTREE(T)$، (k,l)–هسته‌ی درخت T را پیدا می‌کند. این زیربرنامه در فصل ۲ و [۴۸] ارائه شده است. در این زیربرنامه رأس مرکزی v از درخت T، متناظر با مرکز درخت بدون وزن T است. رأس مرکزی رأسی است که با حذف آن بیشترین تعداد رئوس زیردرخت‌های به دست آمده از حذف آن کمینه شود. زیربرنامه‌ی $BEST-TREE(T', v)$ که توسط بکر و همکارانش در [۸] ارائه شده است بهترین زیردرخت S' شامل v را در T'^v با حداکثر k برگ و با قطر حداکثر l پیدا می‌کند. این زیربرنامه نیز در فصل ۲ آمده است.

قضیه ۱.۴.۴. مسأله‌ی پیداکردن یک ۲ – (k,l)–هسته‌ی درخت T می‌تواند در زمان $O(n^۳logn)$ حل شود.

برهان. برای پیداکردن یک ۲ – (k,l)–هسته‌ی درخت T، یالی از آن را حذف و T را به دو قسمت $T_۱$ و $T_۲$ تقسیم می‌کنیم. سپس (k,l)–هسته‌ی هر یک از $T_۱$ و $T_۲$ را پیدا کرده و این زیربرنامه را برای هر یال از T تکرار می‌کنیم. در نهایت بهترین جفت از (k,l)–هسته‌ها را به عنوان ۲ – (k,l)–هسته در نظر می‌گیریم. از طرفی برای پیداکردن (k,l)–هسته‌ی هر یک از $T_۱$ و $T_۲$، از الگوریتم ارائه شده در فصل دو که در [۴۸] نیز آمده است، استفاده می‌کنیم که پیچیدگی این الگوریتم $O(n^۲logn)$ است. همچنین چون در الگوریتم ۲ – (k,l)–هسته تمام یال‌ها یک به یک باید حذف شوند و الگوریتم (k,l)–هسته برای هر زیردرخت به دست آمده از حذف یال تکرار شود، بنابراین پیچیدگی این الگوریتم $O(n^۳logn)$ است. □

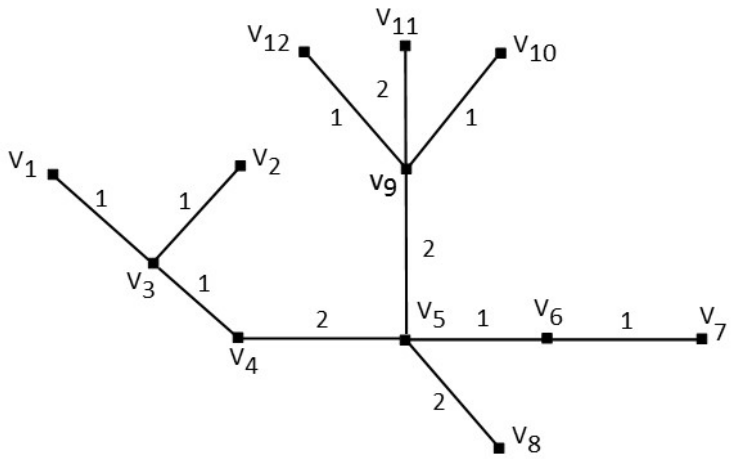
۵.۴ مثال‌های عددی

در این بخش الگوریتم را برای حالات مختلف مجموع وزن‌های رئوس یک درخت استفاده می‌کنیم.

مثال ۱.۵.۴. درخت نشان داده شده در شکل ۱.۴ را در نظر بگیرید. طول هر یال روی آن نوشته و وزن‌های رئوس در جدول ۱.۴ آورده شده است. وزن کل درخت $w(T) = ۳ > ۰$ می‌باشد.
می‌خواهیم ۲ – $(۳,۳)$–هسته‌ی این درخت را پیدا کنیم. از آن جا که $w(T) > ۰$ است طبق الگوریتم باید یالی از T را حذف کنیم و دو زیر درخت $T_۱$ و $T_۲$ را به دست آوریم. فرض کنید یال $(v_۹, v_{۱۰})$ را حذف کنیم. دو زیردرخت $T_۱$ و $T_۲$ به دست آمده از حذف یال (v_j, v_r) مطابق شکل ۲.۴ است. اکنون باید $(۳,۳)$–هسته‌ی هر زیردرخت را پیدا کنیم. چون $T_۱$ فقط رأس $v_{۱۰}$ را دارد، بنابراین $(۳,۳)$–هسته‌ی آن $v_{۱۰}$ است. برای پیداکردن $(۳,۳)$–هسته‌ی زیردرخت $T_۲$ طبق الگوریتم باید رأس مرکزی آن را بیابیم. $v_۵$ را به عنوان رأس مرکزی در نظر می‌گیریم. سپس باید همه‌ی مسیرهایی را که از $v_۵$ شروع می‌شوند و طول آن‌ها حداکثر ۳ است را پیدا کنیم. این مسیرها در شکل ۳.۴ نشان داده شده است.

$w_۱$	$w_۲$	$w_۳$	$w_۴$	$w_۵$	$w_۶$	$w_۷$	$w_۸$	$w_۹$	$w_{۱۰}$	$w_{۱۱}$	$w_{۱۲}$
۳	−۲	۱	−۱	۱	۲	−۳	۲	۲	−۱	−۲	۱

جدول ۱.۴: وزن‌های رئوس درخت شکل ۱.۴ برای مثال ۱.۵.۴.

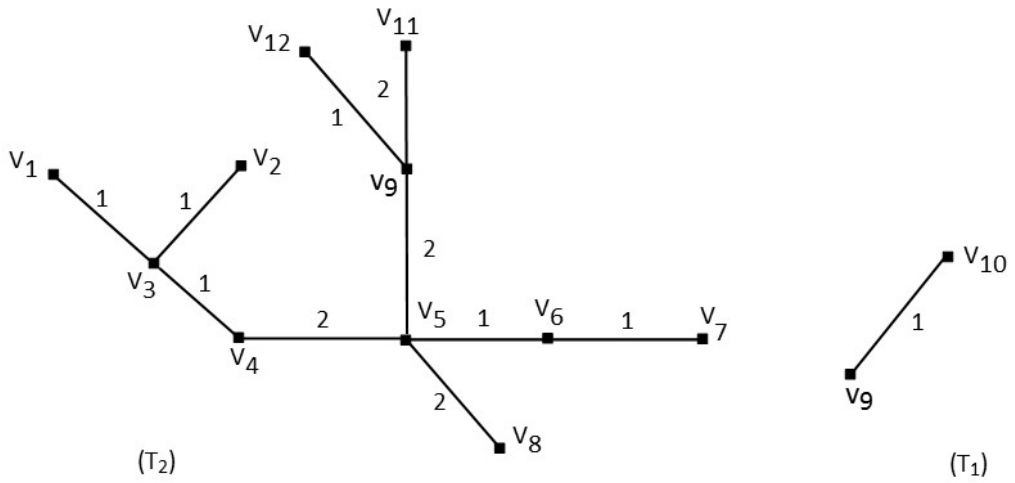


شکل ۱.۴: یک درخت با ۱۲ رأس.

اکنون برای هر مسیر درخت T را هرس می‌کنیم و گام‌های الگوریتم را ادامه می‌دهیم. برای مثال مسیر شماره‌ی ۸ را در شکل ۳.۴ در نظر بگیرید. درخت جدید بعد از هرس در شکل ۴.۴ ارائه شده است. فاصله‌های ذخیره شده‌ی مسیرهای $p : v_۵\_v_۴$ و $p : v_۵\_v_۶$ به ترتیب $sav(v_۵, p_{v_۵v_۴}) = -۲$ و $sav(v_۵, p_{v_۵v_۶}) = +۲$ می‌باشد.
از آن جا که $deg(v_۵) = ۴$ بنابراین $p' = v_۵ - v_۶$ را در نظر می‌گیریم. به طوری که $p'' = p \cup p'$. شکل ۴.۴ را ببینید. همچنین از آن جا که $\hat{T}'^v$ دو برگ دارد طبق الگوریتم تمام مسیرها را در LS نظر می‌گیریم. بنابراین $S' = \hat{T}'^v$ و $d(S') = ۰$ بعد از اجرای گام‌های فوق برای هر مسیر، زیردرخت‌های به دست آمده از حذف $v_۵$ را یکی یکی در نظر می‌گیریم و رأس مرکزی هر یک را پیدا کرده و گام‌های ذکر شده در بالا را برای هر زیردرخت به دست آمده از حذف رأس مرکزی ادامه می‌دهیم. بعد از آن یال‌های دیگر درخت را از قبیل $v_۹ - v_{۱۰}$ یکی یکی حذف می‌کنیم و همه‌ی گام‌های فوق را بعد از حذف هر یال تکرار می‌کنیم. در نهایت $۲ - (۳,۳)$–هسته‌ی درخت T، $(v_۱, v_۸)$ است که با حذف یال $v_۱ - v_۳$ به دست می‌آید و مقدار تابع هدف آن به صورت زیر است:

$$F(T_۱, T_۲) = F(v_۱) + F(v_۸) = ۰ + (-۱۹) = -۱۹$$

مثال ۲.۵.۴. درخت نشان داده شده در شکل ۵.۴ را در نظر بگیرید. طول هر یال روی آن نوشته و وزن‌های رئوس در جدول ۲.۴ آورده شده است. وزن کل درخت $w(T) = -۱ < ۰$ است. می‌خواهیم $۲ - (۳,۴)$–هسته‌ی این درخت را پیدا کنیم. چون $w(T) < ۰$ است مطابق با قضیه‌ی ۲.۳.۲، (k,l)–هسته‌ی یک درخت با وزن منفی یک رأس است. (k,l)–هسته‌ی T، $v_۳$ است که ۱–میانه‌ی درخت T نیز می‌باشد. بهترین



شکل ۲.۴: زیردرخت‌های $T_۱$ و $T_۲$.

جواب از حذف یال $v_۸ - v_۶$ به دست می‌آید. بنابراین $T'_۱ = v_۳$ و $T'_۲ = v_۸$ و مقدار تابع هدف به صورت زیر است:

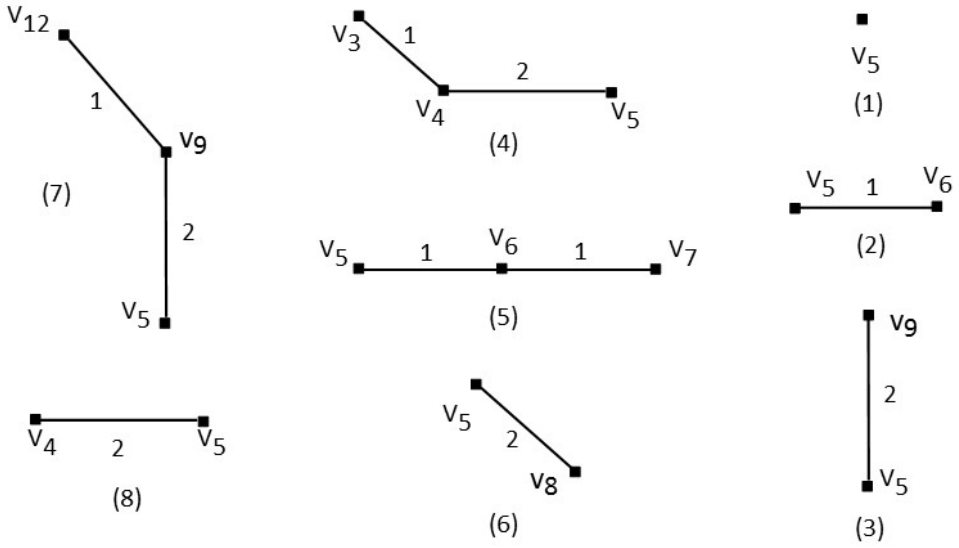
$$F(T'_۱, T'_۲) = F(T'_۱) + F(T'_۲) = -۲۵ + ۰ = -۲۵.$$

$w_۸$	$w_۷$	$w_۶$	$w_۵$	$w_۴$	$w_۳$	$w_۲$	$w_۱$
۱	−۲	−۳	۱	−۲	۲	۳	−۱

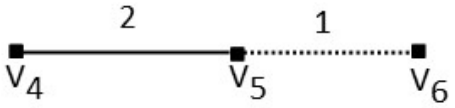
جدول ۲.۴: وزن‌های رئوس درخت شکل ۵.۴ برای مثال ۲.۵.۴.

۶.۴ نتیجه‌گیری

در این فصل مسأله‌ی پیداکردن ۲−(k,l)−هسته‌ی یک درخت در حالت نیمه‌ناخوشایند مورد بررسی قرار گرفت. مدل مسأله بیان و نشان داده شد، برای درخت در حالت بدون وزن جوابی وجود دارد که هیچ یک از دو زیردرخت ۲ − (k,l)−هسته یک رأس نیستند. سپس نشان داده شد در حالتی که مجموع وزن‌های رئوس منفی هستند، ۲ − (k,l)−هسته‌ی T می‌تواند با حذف‌کردن یالی که متصل به یک برگ است به دست آید. در پایان یک الگوریتم برای این مسأله با وزن مثبت/منفی که تعمیمی از الگوریتم ارائه‌شده توسط بکر در [۸] است، ارائه شد. در مطالعات بعدی می‌توان معکوس این مسأله را در نظر گرفت. به این صورت که ۲ − (k,l)−هسته‌ی درخت معلوم و مکان

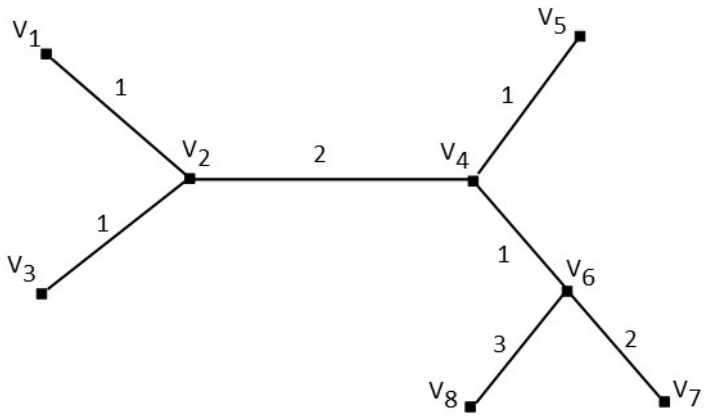


شکل ۳.۴: مسیرهایی که از v_5 شروع می‌شوند با طول حداکثر ۳.



شکل ۴.۴: مسیر شماره ۸ بعد از هرس.

مشتری‌ها مجهول است. به عبارت دیگر هدف در این مسأله پیداکردن مکان رئوس است به طوری که، مجموع فاصله‌ها از این رئوس تا (k,l) – ۲–هسته کمینه باشد.



شکل ۵.۴: یک درخت با ۸ رأس.

فصل ۵

هسته‌ی درخت‌های بازه‌ای وزن‌دار

۱.۵ مقدمه

گراف‌های بازه‌ای یک کلاس از محبوب‌ترین و قابل فهم‌ترین گراف‌ها هستند. این گراف‌ها در زمینه‌های مختلف کاربرد دارند و اغلب می‌توان آن‌ها را توسط نظریه‌ی گراف کلاسیک مدل‌بندی کرد. به عنوان مثال مسائل زمان‌بندی و انبارداری به مسأله‌ی پیداکردن کمترین رنگ‌بندی و مسائل پوشش گروه به گراف‌های بازه‌ای مناسب تبدیل می‌شوند [۳۱]. از طرف دیگر مسائل $NP-$سخت کلاسیک زیادی وجود دارند که با تبدیل‌شدن به گراف‌های بازه‌ای قابل حل هستند. به عنوان مثال مسائل مجموعه مستقل[۱] و مجموعه غالب[۲] و خوشه[۳] با تبدیل‌شدن به گراف‌های بازه‌ای در زمان چندجمله‌ای حل می‌شوند [۲۴] و [۲۸]. معمولاً بیشتر مسائل کاربردی به گراف‌های بازه‌ای چندگانه تبدیل می‌شوند. یک گراف بازه‌ای چندگانه تعداد متناهی از بازه‌های متقاطع روی خط حقیقی است. گراف‌های متقاطع خانواده‌ای از این گراف‌های بازه‌ای هستند.

---

[۱] *Independent Set*

[۲] *Dominating Set*

[۳] *Clique*

در سال ۲۰۰۴، اوهرا[۴] و اونو[۵] [۶۱] زیرکلاسی از گراف‌های بازه‌ای را به نام گراف‌های بازه‌ای محدب دوطرفه معرفی کردند. آن‌ها مسأله‌ی پیداکردن طولانی‌ترین مسیر یک گراف بازه‌ای را در این کلاس از گراف‌ها در زمان $o(n^3(m+nlogn))$ حل کردند و مسأله را برای حالت کلی گراف‌ها باز گذاشتند. اوآننیدو[۶] و همکاران در سال ۲۰۱۰ با استفاده از روش برنامه‌ریزی پویا این مسأله را برای کلیه‌ی گراف‌های بازه‌ای در زمان $o(n^4)$ حل کردند [۴۰].

در این فصل ما به بررسی مسأله‌ی هسته بر روی گراف‌های بازه‌ای می‌پردازیم. یک گراف بازه‌ای مجموعه‌ای از بازه‌ها است که این بازه‌ها می‌توانند متناظر با رئوس یک گراف باشند به طوری که اگر یالی بین رأس v_i و v_j برقرار باشد، دو بازه متناظر با آن‌ها I_i و I_j دارای اشتراک هستند.

گراف‌های بازه‌ای ساختار توپولوژی مناسبی دارند. همچنین ابزار مهمی در زمینه‌های کاربردی از جمله باستان‌شناسی، زیست‌شناسی، زمان‌بندی و کنترل‌ترافیک هستند [۳۵] و [۴۱]. به همین دلیل محققان زیادی علاقه‌مند به مطالعه بر روی این گراف‌ها شدند (مراجع [۱۱] و [۱۷] و [۱] را ببینید). مسأله‌ی ۱−میانه روی گراف‌های بازه‌ای با وزن مثبت توسط بسپامیاتنیخ[۷] و همکارانش [۱۱] بررسی شد. آن‌ها الگوریتمی خطی برای حل این مسأله پیشنهاد کردند . چنگ[۸] و کانگ[۹] [۱۸] به بررسی مساله $p-$ ماکسین روی گراف‌های بازه‌ای پرداخته‌اند. قاسمی نیز در [۲] مسأله‌ی $p-$ ماکسین روی گراف‌های بازه‌ای که هر بازه دارای یک وزن مثبت است را بررسی کرده است.

درخت‌های بازه‌ای حالت خاصی از گراف‌های بازه‌ای هستند که از هر بازه به بازه‌ی دیگر فقط یک مسیر وجود دارد و این گراف فاقد دور است. ساختار داده‌ای درخت‌های بازه‌ای توسط ربیعی در [۳] بررسی شده‌است.

ما در این فصل ابتدا هسته را در یک گراف بازه‌ای تعریف کرده، سپس با ارائه خواصی برای این مساله روی درخت‌های بازه‌ای، الگوریتمی خطی با پیچیدگی زمانی $O(n)$ برای پیداکردن هسته‌ی درخت بازه‌ای ارائه می‌کنیم.

۲.۵ ویژگی‌های ساختاری گراف‌های بازه‌ای

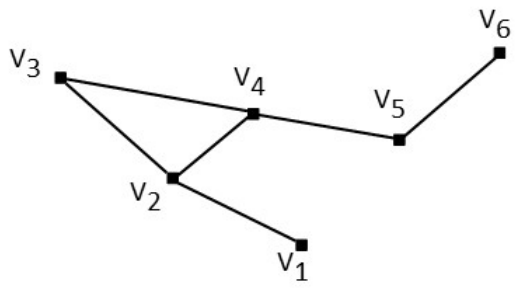
گراف G یک گراف بازه‌ای است اگر رئوس آن با خانواده‌ای از بازه‌ها مانند F روی یک خط حقیقی یک به یک متناظر باشند. همچنین بازه‌های متناظر با دو رأس همسایه

---

[۴] *Uehara*
[۵] *Uno*
[۶] *Ioannidou*
[۷] *Bespamyatnikh*
[۸] *Cheng*
[۹] *Kang*

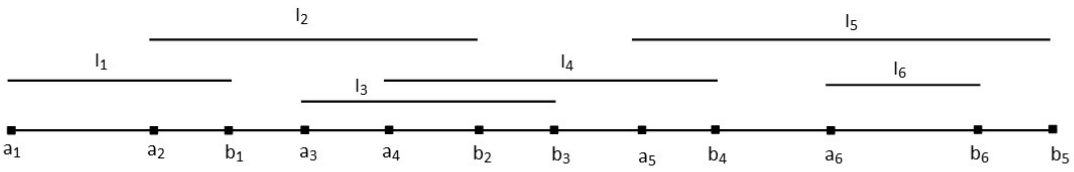
در G متقاطع هستند. به عبارت دیگر اگر بین دو رأس v_i و v_j در یک گراف بازه‌ای یالی وجود داشت، بازه‌های متناظر با آن مانند $I_i = [a_i, b_i]$ و $I_j = [a_j, b_j]$ روی خط حقیقی دارای اشتراک هستند. F مدل گسترده‌ی G نامیده می‌شود [۴]. برای درک بهتر مثال‌های زیر را ببینید.

مثال ۱.۲.۵. فرض کنید گراف نشان داده شده در شکل ۱.۵ گراف G باشد.



شکل ۱.۵: گراف بازه‌ای G

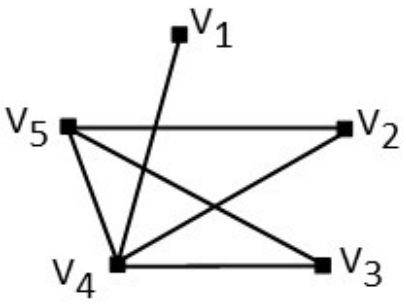
این گراف یک گراف بازه‌ای است زیرا مدل گسترده‌ی آن یا F وجود دارد. یعنی می‌توان روی خط حقیقی متناظر با هر یک از رئوس G مانند v_i بازه‌ی I_i را در نظر گرفت به طوری که اگر یالی بین دو رأس از G وجود داشته باشد بازه‌های متناظر با آن‌ها با هم اشتراک دارند. شکل ۲.۵ F را نشان می‌دهد.



شکل ۲.۵: مدل گسترده‌ی گراف G

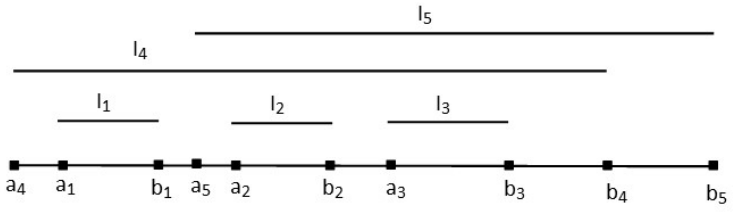
همان‌طور که توضیح داده شد در شکل ۱.۵ می‌بینیم که رأس $v_۴$ با سه رأس $v_۲$، $v_۳$ و $v_۵$ همسایه است. بنابراین بازه‌ی متناظر با رأس $v_۴$ یعنی $[a_۴, b_۴]$ باید با سه بازه‌ی $[a_۲, b_۲]$، $[a_۳, b_۳]$ و $[a_۵, b_۵]$ دارای اشتراک باشد. شکل ۲.۵ این مطلب را نشان می‌دهد.

مثال ۲.۲.۵. فرض کنید گراف نشان داده شده در شکل ۳.۵ گراف G باشد.



شکل ۳.۵: گراف بازه ای G

این گراف نیز مانند گراف G نشان داده شده در مثال قبل یک گراف بازه‌ای است. شکل ۴.۵ مدل گسترده‌ی آن یعنی F را نشان می‌دهد.

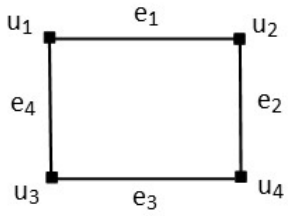


شکل ۴.۵: مدل گسترده ی گراف G

نکته‌ی قابل توجه این که می‌توان گرافی یافت که نتوان آن را به صورت یک گراف بازه‌ای تعریف کرد. برای پاسخ به این سوال که آیا یک گراف می‌تواند یک گراف بازه‌ای باشد یا نه لم زیر را داریم[۱۰].

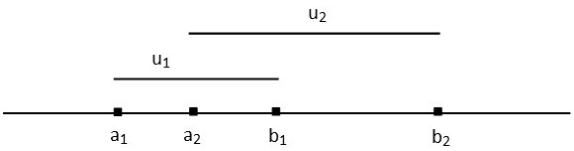
لم ۱.۲.۵. هر گراف که یك گراف القایی ۴ رأسی دوری داشته باشد نمی‌تواند بازه‌ای باشد.
یعنی به بیان ساده‌تر در هر گراف اگر بتوان یك مربع (چهار راسی که با چهار یال متوالیاً به هم وصل شده‌اند) یافت به قسمی که بین چهار رأس آن به جز چهار یال مربع، یال دیگری نباشد آن گراف الزاماً نمی‌تواند بازه‌ای باشد.

برهان. چهار رأس $u_۱$، $u_۲$، $u_۳$ و $u_۴$ را در نظر بگیرید که با یال‌های $e_۱$، $e_۲$، $e_۳$ و $e_۴$ مانند شکل ۵.۵ به هم متصل شده‌اند.



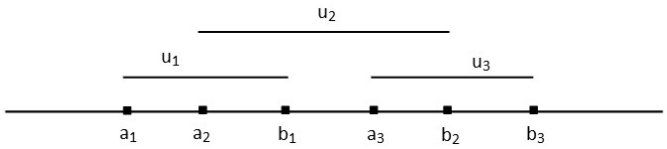
شکل ۵.۵: گراف بازه‌ای G

به برهان خلف اگر این گراف بازه‌ای باشد آن‌گاه بازه‌ی $u_۱$ با $u_۲$ اشتراک دارد. پس شکلی مانند شکل ۶.۵ دارند.



شکل ۶.۵: بازه‌های $u_۱$ و $u_۲$

به طریق مشابه $u_۳$ هم با $u_۲$ اشتراک دارد(به خاطر وجود یال $e_۲ = u_۲u_۳$) و از طرفی چون با $u_۱$ یالی ندارد پس $u_۳$ با $u_۱$ اشتراکی ندارد. پس تنها حالتی که بتوان برای $u_۳$ بازه‌ای فرض کرد که با بازه‌ی $u_۲$ اشتراک داشته و با $u_۱$ نداشته باشد به صورت شکل ۷.۵ است.



شکل ۷.۵: بازه‌های $u_۱$ و $u_۲$ و $u_۳$

حال نوبت پیداکردن بازه‌ی $u_۴$ است. دقت کنید $u_۴$ هم به $u_۱$ و هم به $u_۳$ یال دارد ولی به $u_۲$ یالی ندارد. پس باید بتوان در شکل بالا بازه‌ای برای آن یافت که هم با بازه‌ی $u_۳$ اشتراک داشته باشد و هم با بازه‌ی $u_۱$ اشتراک داشته باشد و از طرفی از محدوده‌ی بازه هم نگذرد. که واضح است چنین چیزی ممکن نیست. لذا فرض خلف باطل است و ثابت می‌گردد که ۴ رأس $u_۱$، $u_۲$، $u_۳$ و $u_۴$ نمی‌توانند بازه‌های مناسب اختیار کنند. □

۳.۵ مفاهیم و تعاریف

قبل از بیان مسأله‌ی میانه‌ی بازه‌ای به ذکر مفاهیم و تعاریف زیر می‌پردازیم.

فرض کنید $G = (V, E)$ یک گراف با مجموعه رئوس V و مجموعه یال‌های E باشد. در یک گراف بازه‌ای هر رأس v_i تناظر یک به یک با بازه I_i دارد و یال در صورتی وجود دارد که $I_i \cap I_j \neq \emptyset$. فرض کنید مجموعه‌ی $I = \{I_۱, I_۲, \ldots, I_n\}$ مجموعه‌ی بازه‌های متناظر با رئوس در گراف G باشد. طول هر یال را یک در نظر گرفته و اگر رأس v_i دارای وزنw_i باشد طول بازه I_i را w_i در نظر می‌گیریم. بنابراین یک گراف بازه‌ای مجموعه‌ای از بازه‌ها است که اگر دو بازه I_i و I_j با هم اشتراک داشته باشند، یالی بین رأس v_i و v_j برقرار است. همچنین فرض کنید $C(I_i, I_j)$ کوتاه‌ترین مسیر بازه‌های متصل I_i و I_j باشد. تعداد یال‌های روی $C(I_i, I_j)$ را فاصله بین I_i و I_j در نظر گرفته و با $d(I_i, I_j)$ نشان می‌دهیم.

اگر $I_i = [a_i, b_i]$ باشد که a_i و b_i به ترتیب نقاط انتهایی چپ و راست بازه I_i هستند. آن‌گاه هیچ دو بازه‌ای در مجموعه بازه‌های I نقاط انتهایی یکسانی ندارند و $a_i < b_i$.

برای راحتی کار با بازه‌ها برداری شامل نقاط پایانی بازه‌ها را تشکیل می‌دهیم. به این منظور نقاط انتهایی بازه‌ها را مرتب کرده و در بردارL قرار می‌دهیم. بنابراین

$$L = \{p_۱, p_۲, \ldots, p_{۲n}\}$$

که p_i ها در آن یا a_i و یا b_i هستند. در این بردار اگر $i < j$ آن گاه $b_i < b_j$ و اگر انتهای بازه‌ی I_i ابتدای بازه‌ی I_j باشد آن گاه در بردار L، b_i قبل از a_j قرار می‌گیرد. چنگ و کانگ [۱۸] نشان دادند که این برچسب‌گذاری در زمان$O(nlogn)$ به دست می‌آید.
اگر s کوچکترین نقطه پایانی چپ و t بزرگترین نقطه پایانی راست باشد، قرار می‌دهیم $t = b_n$، $s = a_l$ و $l \in \{1, \ldots, n\}$. همچنین فرض می‌کنیم $\cup_{i=1}^{n} I_i = [s, t]$. بنابراین اجتماع بازه‌های I، بازه‌ی متصل $[s, t]$ است.
نکته‌ی قابل توجه این است که با استفاده از این برچسب‌گذاری می‌توان به آسانی لیستی از تمام بازه‌های I که توسط a'_i ها یا b'_i ها مرتب شده‌اند به دست آورد.

تعریف ۱.۳.۵. بازه‌ی I_i را در I ماکسیمال گوئیم، هر گاه مشمول هیچ بازه‌ی دیگری نباشد.

تعریف ۲.۳.۵. بازه‌ی I_i را در I مینیمال گوئیم، هر گاه شامل هیچ بازه‌ی دیگری نباشد و بازه‌ی ماکسیمالی مانند I_j وجود داشته باشد که شامل آن باشد.

ما همچنین دو بازه‌ی I_i و I_j را که با هم دارای اشتراک هستند با $I_i \sim I_j$ و اگر هیچ اشتراکی نداشته باشند با $I_i \nsim I_j$ نشان می‌دهیم.
اگر $q \in L$ یک نقطه پایانی ازبازه‌ی I_i باشد. تعاریف زیر را در نظر بگیرید،

$$w_{aL}(q) = \sum_{I_j \in I | a_j < q} w(I_j) \qquad w_{bL}(q) = \sum_{I_j \in I | b_j < q} w(I_j)$$

$$w_{aR}(q) = \sum_{I_j \in I | a_j > q} w(I_j) \qquad w_{bR}(q) = \sum_{I_j \in I | b_j > q} w(I_j)$$

همچنین توابع $F_{aL}(q)$ و $F_{aR}(q)$ را به صورت زیر تعریف می‌کنیم:

$$F_{aL}(q) = \sum_{I_j \in I | a_j < q} w(I_j) d(I_j, I_i)$$

$$F_{aR}(q) = \sum_{I_j \in I | a_j > q} w(I_j) d(I_j, I_i)$$

برای هر نقطه‌ی q روی خط تعریف می‌کنیم،

$$RSUC(q) = I_i = [a_i, b_i] \in I$$

اگر و تنها اگر $b_i = max\{b_j \mid q \notin I_j\}$ (یعنی I_i بازه‌ای است که با بازه‌ی شامل q اشتراک دارد و کران بالای آن بزرگترین است) و

$$LSUC(q) = I_i \in I$$

اگر و تنها اگر $a_i = min\{a_j \mid q \notin I_j\}$ (یعنی I_i بازه‌ای است که با بازه‌ی شامل q اشتراک دارد و کران پایین آن کوچکترین است).
برای یک بازه مانند I_i تعریف می‌کنیم،

$$RSUC(I_i) = RSUC(b_i)$$

$$LSUC(I_i) = LSUC(a_i)$$

و چون $\bigcup_{i=۱}^{n} I_i$ یک مولفه‌ی پیوسته است، اگر $i = n$ آن گاه

$$RSUC(I_i) = I_i$$

و اگر I_i بازه‌ای با کوچکترین نقطه‌ی پایانی چپ باشد آن گاه

$$LSUC(I_i) = I_i$$

$RSUC(q, i)$ برابر است با i امین $RSUC$ برای نقطه q که به صورت زیر تعریف می‌شود:

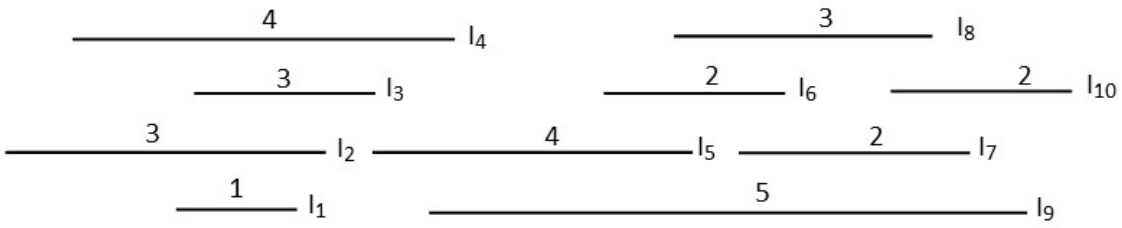
$$RSUC(RSUC(\cdots RSUC(q, ۰)))$$

که در آن $RSUC$، $i+۱$ بار ظاهر می‌شود و $RSUC(q, ۰) = q$ به طور مشابه $LSUC(q, i)$ نیز برای هر بازه‌ی I_i تعریف می‌شود. همچنین

$$RSUC(I_j, i) = RSUC(b_j, i)$$

$$LSUC(I_j, i) = LSUC(a_j, i)$$

مثال ۱.۳.۵. برای درک بیشتر مفاهیم فوق شکل ۸.۵ را در نظر بگیرید.



شکل ۸.۵: بازه‌های مربوط به مثال ۱.۳.۵

با توجه به شکل ۸.۵ داریم:

$$RSUC(I_۱) = I_۴ \qquad RSUC(I_۴) = I_۹ \qquad RSUC(I_۹) = I_{۱۰}$$

$$LSUC(I_{۱۰}) = I_۹ \qquad LSUC(I_۹) = I_۴ \qquad LSUC(I_۴) = I_۲$$

$$RSUC(I_۱, ۳) = I_{۱۰} \qquad LSUC(I_{۱۰}, ۳) = I_۲$$

۴.۵ مسأله‌ی میانه بازه‌ای

مسأله‌ی ۱−میانه در یک گراف بازه‌ای عبارت است از یافتن یک بازه یا سرویس‌دهنده روی شبکه با n بازه به عنوان نقاط تقاضا، به طوری که مجموع فاصله‌های $n-1$ بازه‌ی دیگر تا بازه مورد نظر کمینه شود. به عبارت دیگر در این مسأله می‌خواهیم بازه I_i را که $i \in \{1,\ldots,n\}$ بیابیم به طوری که تابع هدف زیر را کمینه کند:

$$F(I_i) = \sum_{k=1}^{n} w(I_k) min\ d(I_k, I_i)$$

بنابراین همان‌طور که در [۱۸] آمده‌است، برای یک کاندید ۱−میانه تابع هدف می‌تواند به صورت زیر فرمول‌بندی شود:

$$F(I_i) = \sum_{k=1}^{n} w(I_k) d(I_k, I_i) = F_{aL}(a_i) + F_{aR}(a_i)$$

به طوری که بازه‌ای که تابع فوق را کمینه کند ۱−میانه است. همچنین یانگ چنگ در [۱۸] نشان داده است که مسأله‌ی ۱−ماکسین و ۱−میانه روی گراف‌های بازه‌ای که نقاط پایانی آن‌ها مرتب شده‌اند در زمان $O(n)$ قابل حل هستند.
مسأله p−میانه در یک گراف بازه‌ای عبارت است از یافتن p بازه یا سرویس‌دهنده روی شبکه با n بازه به عنوان نقاط تقاضا، به طوری که مجموع فاصله‌های $n-p$ بازه‌ی دیگر تا p بازه مورد نظر کمینه شود. بنابر تعاریف فوق مسأله p−میانه در گراف‌های بازه‌ای عبارت است از پیداکردن مجموعه‌ی $I = \{I_{i_1}, I_{i_2}, \ldots, I_{i_p}\}$ به طوری‌که تابع هدف زیر کمینه شود:

$$F(I) = \sum_{k=1}^{n} w(I_k) min_{1 \leqslant j \leqslant p} d(I_k, I_{i_j})$$

به طور مستقیم از تعریف $RSUC$ می‌توان نتایج زیر را به دست آورد:

نتیجه ۱.۴.۵. برای هر دو بازه‌ی I_i و I_j در I با $i < j$ و $b_i < b_j$ کوتاه‌ترین مسیر با طول d به صورت زیر وجود دارد:

$$p(I_i, I_j) = \{I_i, RSUC(I_i, 1), \ldots, RSUC(I_i, d-1), I_j\}$$

نتیجه ۲.۴.۵. برای هر دو بازه‌ی I_i و I_j در I با $a_i < a_j$ کوتاه‌ترین مسیر با طول d به صورت زیر وجود دارد:

$$p(I_i, I_j) = \{I_j, RSUC(I_j, 1), \ldots, RSUC(I_j, d-1), I_i\}$$

چن [۱۸] با استفاده از نتایج فوق لم‌های مهمی را برای به دست آوردن فاصله‌ی بین بازه‌ها به صورت زیر ارائه کرده است،

لم ۱.۴.۵. ۱) اگر سه بازه‌ی I_h، I_i و I_j در I وجود داشته باشند به طوری که $b_h < b_j$، $b_h < b_i$ و $a_i < a_j$ باشد، آن‌گاه $d(I_h, I_i) \leqslant d(I_h, I_j)$.
۲) اگر سه بازه‌ی I_h، I_i و I_j در I وجود داشته باشند به طوری که $a_h > a_j$، $a_h > a_i$ و $b_i < b_j$ باشد، آن گاه $d(I_h, I_j) \leqslant d(I_h, I_i)$.

برهان. برای اثبات قسمت ۱ با استفاده از نتیجه‌ی ۱.۴.۵ دو عدد صحیح k و l وجود دارد به طوری که

$$p(I_h, I_i) = \{I_h, RSUC(I_h, ۱), \ldots, RSUC(I_h, k-۱), I_i\}$$
$$p(I_h, I_j) = \{I_h, RSUC(I_h, ۱), \ldots, RSUC(I_h, l-۱), I_j\}$$

چون $a_i < a_j$ واضح است که $k \leqslant l$، بنابراین $d(I_h, I_i) \leqslant d(I_h, I_j)$.
قسمت ۲ نیز به طور مشابه اثبات می‌شود. □

لم ۲.۴.۵. اگر بازه‌ی I_i در بازه‌ی I_j قرار داشته باشد یعنی $a_j < a_i < b_i < b_j$ آن‌گاه برای هر بازه‌ی دیگر مانند $I_h(I_h \neq I_i, I_j)$داریم:

$$d(I_h, I_j) \leqslant d(I_h, I_i)$$

برهان. برای اثبات این لم سه حالت زیر را در نظر می‌گیریم:

- حالت اول: اگر $I_h \sim I_j$ آن‌گاه $d(I_h, I_j) = ۱ \leqslant d(I_h, I_i)$.
- حالت دوم: اگر $I_h \nsim I_j$ و $b_h < a_j$ آن‌گاه واضح است که $b_h < b_i$، $b_h < b_j$ و $a_j < a_i$. از قسمت اول لم ۱.۴.۵ داریم: $d(I_h, I_j) \leqslant d(I_h, I_i)$
- حالت سوم: اگر $I_h \nsim I_j$ و $a_h > b_j$ آن‌گاه واضح است که $a_h > a_i$، $a_h > a_j$ و $b_i < b_j$. از قسمت دوم لم ۱.۴.۵ داریم: $d(I_h, I_j) \leqslant d(I_h, I_i)$

□

قضیه ۱.۴.۵. هیچ بازه‌ی مینیمالی نمی‌تواند کاندیدی برای جواب مسأله ۱–میانه باشد.

برهان. با توجه به لم ۲.۴.۵ اگر بازه‌ی I_i در بازه‌ی I_j قرار داشته باشد یا $a_j < a_i < b_i < b_j$ آن‌گاه برای هر بازه‌ی دیگر $I_h(I_h \neq I_i, I_j)$ داریم: $d(I_h, I_j) \leqslant d(I_h, I_i)$ بنابراین چون فاصله بازه‌های دیگر به I_j کمتر از I_i است و برای مینیمم‌شدن تابع $F(I)$ کمترین فاصله لازم است. بازه‌ی I_j می‌تواند جایگزین بازه I_i شود. بنابراین بازه I_i نمی‌تواند به عنوان کاندیدی برای مسأله‌ی ۱–میانه درنظر گرفته شود. □

۵.۵ مسأله‌ی پیداکردن هسته‌ی یک درخت بازه‌ای

هسته‌ی یک گراف مسیری است در گراف که مجموع فاصله‌ها از تمام رئوس تا این مسیر کمینه شود. بنابراین در یک گراف بازه‌ای یک هسته عبارت است از مجموعه‌ای از بازه‌های متصل به هم مانند $I = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ که مجموع فاصله‌ها از تمام بازه‌ها به این مسیر بازه‌ای کمینه شود. لذا با توجه به تعاریف فوق تابع هدف در این مسأله را می‌توان به صورت زیر در نظر گرفت:

$$F(I) = \sum_{k=۱}^{n} w(I_k) min_{۱\leqslant j\leqslant p} d(I_k, I_{i_j})$$

به طوری که $\forall u \in \{۱, ۲, \ldots, p-۱\}$ و $I_{i_u} \cap I_{i_{u+۱}} \neq \emptyset$. در ادامه به بررسی مساله پیداکردن هسته روی یک درخت بازه‌ای می‌پردازیم.

قضیه ۱.۵.۵. در یک درخت بازه‌ای اگر دو بازه‌ی I_i و I_j که $a_i \leq a_j$ و $b_i \leq b_j$ با هم اشتراک داشته و یالی بین رئوس متناظر با آن‌ها موجود باشد، هیچ بازه‌ی $I_h \neq I_i, I_j$ دیگری نمی‌تواند وجود داشته باشد به طوری که

$$a_j \leq a_h < b_h \leq b_i$$

برهان. فرض کنید بازه‌ی I_h ی وجود دارد که $I_h \neq I_i, I_j$ و $a_j \leq a_h < b_h \leq b_i$. از آن جا که $a_j \leq a_h$ و $a_h < b_h \leq b_i$ و $b_i \leq b_j$ بنابراین $a_j < a_h \leq b_j$ یعنی بازه‌ی I_h با I_j دارای اشتراک است و $I_j \cap I_h \neq \emptyset$. همچنین چون $a_h \leq a_j$ و $a_i \leq a_j$ بنابراین $a_i \leq a_h$ است و چون $a_h < b_h \leq b_i$ لذا $b_h \leq b_i$ در نتیجه $a_i < b_h \leq b_i$ یعنی $I_i \cap I_h \neq \emptyset$.
از طرفی طبق فرض $I_i \cap I_j \neq \emptyset$ بنابراین $I_i \cap I_h \cap I_j \neq \emptyset$ که نشان می‌دهد گراف دارای دور است و با فرض درخت بودن گراف تناقض دارد. □

قضیه ۲.۵.۵. اگر $I = \{I_۱, I_۲, \ldots, I_n\}$ مجموعه‌ی بازه‌ها در درخت بازه‌ای G و $I_p = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ هسته‌ی درخت G باشد. آن گاه

$$F(I_p) = F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱}) - \sum_{j=۱}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{p} w(I_{i_j})$$

برهان. می‌دانیم $F(I_{i_۱}) = F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱})$ با اضافه کردن بازه‌های $I_{i_۲}, I_{i_۳}, \ldots, I_{i_p}$ به $I_{i_۱}$ مسیر $I_p = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ تشکیل می‌شود. به وضوح $F(I_p) < F(I_{i_۱})$. فرض کنید $E + F(I_p) = F(I_{i_۱})$ که $E > ۰$. ثابت می‌کنیم

$$E = \sum_{j=۱}^{p-۱} w_{aR}(b_{i_j}) + \sum_{j=۲}^{p} w(I_{i_j})$$

برای بازه‌های I_j که $b_{i_۱} < a_j < b_j < b_{i_۲}$، چون بازه‌ی I_j با $I_{i_۱}$ دارای اشتراک نیست و با $I_{i_۲}$ اشتراک دارد، پس $d(I_j, I_{i_۱}) = ۲$ و $d(I_j, I_{i_۲}) = ۱$.

از طرفی مجموع وزن‌های بازه‌های بین $I_{i_۱}$ و $I_{i_۲}$ به اندازه‌ی $w_{aR}(I_{i_۱}) - w_{aR}(I_{i_۲})$ می‌باشد و از آن جا که فاصله تا $I_{i_۱}$، ۲ واحد است که یک واحد از آن فاصله تا $I_{i_۲}$ می‌باشد. بنابراین از مقدار $F(I_{i_۱})$ به اندازه‌ی $w_{aR}(I_{i_۱}) - w_{aR}(I_{i_۲})$ کاسته می‌شود.
همچنین برای بازه‌های I_j که $b_{i_۲} < a_j < b_j < b_{i_۳}$ ، چون بازه‌ی I_j با $I_{i_۲}$ دارای اشتراک نیست و با $I_{i_۳}$ اشتراک دارد، پس $d(I_j, I_{i_۱}) = ۳$ و $d(I_j, I_{i_۳}) = ۱$ و $d(I_{i_۱}, I_{i_۳}) = ۲$.
از طرفی مجموع وزن‌های بازه‌های بین $I_{i_۲}$ و $I_{i_۳}$ به اندازه‌ی $w_{aR}(I_{i_۲}) - w_{aR}(I_{i_۳})$ است و از آن جا که فاصله تا $I_{i_۱}$، ۳ واحد است که یک واحد از آن فاصله تا $I_{i_۳}$ می‌باشد و $d(I_{i_۱}, I_{i_۳}) = ۲$. بنابراین $F(I_{i_۱})$ به اندازه‌ی $۲[w_{aR}(I_{i_۲}) - w_{aR}(I_{i_۳})]$ کاهش می‌یابد.
با ادامه این روند برای بازه‌های I_j که $b_{i_{p-۱}} < a_j < b_j < b_{i_p}$ ، داریم
$d(I_j, I_{i_۱}) = p$ و $d(I_j, I_{i_k}) = ۱$ که به اندازه‌ی $(p-۱)[w_{aR}(I_{i_{p-۱}}) - w_{aR}(I_{i_p})]$ و برای بازه‌های I_j که $b_{i_p} < a_j$، اگر $d(I_j, I_{i_p}) = k$ ، چون $d(I_j, I_{i_۱}) = p + k - ۱$ بنابراین به اندازه‌ی $(p-۱)w_{aR}(I_{i_p})$ از مقدار $F(I_{i_۱})$ کم می‌شود.
از طرف دیگر خود بازه‌های $I_{i_۲}, I_{i_۳}, \ldots, I_{i_p}$ که در مسیر قرار گرفته‌اند نیز مقدار $F(I_{i_۱})$ را کاهش می‌دهند این بازه‌ها در محاسبات قبل به طور کامل محاسبه نمی‌شوند. مثلاً بازه‌ی I_j در $d(I_j, I_{i_۱}) = j-۱$ در نظر گرفته می‌شود اما $(j-۲)[w_{aR}(I_{i_{j-۲}}) - w_{aR}(I_{i_{j-۱}})]$
بنابراین برای هر بازه‌ی $j = ۲,۳,\ldots,p$، به اندازه‌ی $w_{aR}(I_{i_j})$ باید از مقدار $F(I_p)$ کم کنیم. لذا داریم:

$$F(I_p) = F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱}) - [w_{aR}(b_{i_۱}) - w_{aR}(b_{i_۲})] - ۲[w_{aR}(b_{i_۲}) - w_{aR}(b_{i_۳})] - \ldots -$$
$$(p-۱)[w_{aR}(b_{i_{p-۱}}) - w_{aR}(b_{i_p})] - pw_{aR}(b_{i_p}) =$$
$$F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱}) - \sum_{j=۱}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{p} w(I_{i_j})$$

□

مثال زیر کاربرد قضیه‌ی ۲.۵.۵ را نشان می‌دهد.

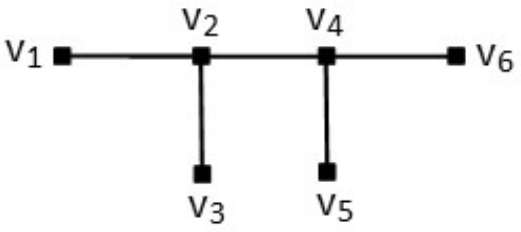
مثال ۱.۵.۵. گراف نشان داده شده در شکل ۹.۵ همراه با وزن رئوسی که در جدول ۱.۵ آمده است را در نظر بگیرید. بازه‌ای متناظر با گراف شکل ۹.۵ در شکل ۱۰.۵ آمده است.

$w_۱$	$w_۲$	$w_۳$	$w_۴$	$w_۵$	$w_۶$
۲	۲	۱	۲	۱	۲

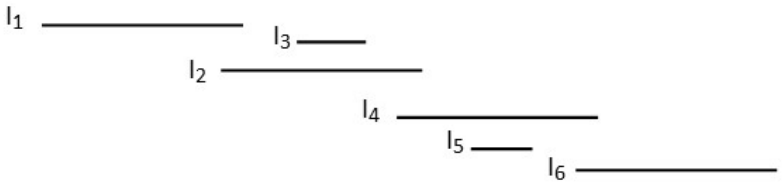
جدول ۱.۵: وزن‌های رئوس درخت شکل ۹.۵ در مثال ۱.۵.۵.

برای بازه‌ی $I_۲$، مقدار تابع هدف به صورت زیر محاسبه می‌شود.

$$F(I_۲) = F_{aL}(a_۲) + F_{aR}(a_۲) = ۲ + ۹ = ۱۱$$



شکل ۹.۵: گراف بازه‌ای G



شکل ۱۰.۵: بازه‌های متناظر با گراف شکل ۹.۵

حال اگر مسیر $I = \{I_۲, I_۴, I_۵\} = \{I_{i_۱}, I_{i_۲}, I_{i_۳}\}$ را در نظر بگیریم. داریم:

$$F(I) = F_{aL}(a_۲) + F_{aR}(a_۲) - \sum_{j=۱}^{۲} w_{aR}(b_{i_j}) - \sum_{j=۲}^{۳} w(I_{i_j}) = ۱۱ - ۳ - ۳ = ۵$$

بنابراین با استفاده از قضیه‌ی ۲.۵.۵ اگر مقدار تابع هدف را در ابتدای مسیر داشته باشیم با استفاده از مجموع وزن‌ها با محاسبات کمتری می‌توان مقدار تابع هدف را برای مسیر مورد نظر به دست آورد. همچنین با استفاده از این قضیه می‌توان تغییرات تابع هدف را در زمان اضافه و کم شدن بازه‌ها به راحتی به دست آورد.

نتیجه ۱.۵.۵. اگر $I_۱$ بازه‌ای در درخت بازه‌ای G و $F(I_۱) = F_{aL}(a_۱) + F_{aR}(a_۱)$ باشد و $I_p = \{I_{i_۱}, I_{i_۲}\}$ مسیری از درخت G باشد، آن‌گاه

$$F(I_p) = F(I_۱) - w(I_۲)$$

قضیه ۳.۵.۵. اگر $I_p = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ یک هسته‌ی درخت G باشد هیچ یک از بازه‌های $I_{i_۲}, I_{i_۳}, \ldots, I_{i_p}$ بازه مینیمال نیستند.

برهان. طبق قضیه ۱.۴.۵ اگر بازه‌ی I_{i_j}، $j = ۲, \ldots, p$ یک بازه‌ی مینیمال باشد بازه ماکسیمالی مشمول آن مانند I_{i_k} وجود دارد که فاصله‌ی بازه‌های دیگر تا آن از I_{i_j} کمتر است. بنابراین می‌تواند I_{i_k} جایگزین I_{i_j} باشد. □

نتیجه ۲.۵.۵. اگر $I_p = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ یک هسته‌ی درخت G باشد هیچ یک از بازه‌های $I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}$ نمی‌توانند بازه‌ای غیرماکسیمال باشند. به عبارت دیگر تمام بازه‌های روی هسته ماکسیمال هستند.

قضیه ۴.۵.۵. اگر $I = \{I_۱, I_۲, \ldots, I_p\}$ مجموعه‌ی بازه‌ها در درخت بازه‌ای G و $I_p = \{I_{i_۱}, I_{i_۲}, \ldots, I_{i_p}\}$ مسیری از درخت G باشد و $I_k = \{I_{i_k}, I_{i_{k+۱}}, \ldots, I_{i_p}\}$ که $۱ < k < p$ آن‌گاه

$$F(I_k) =$$
$$F(I_p) - F_{aL}(a_{i_۱}) - F_{aR}(a_{i_۱}) + F_{aL}(a_{i_k}) + F_{aR}(a_{i_k}) - \sum_{j=۱}^{k-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{k} w(I_{i_j})$$

برهان. از قضیه ۲.۵.۵ داریم:

$$F(I_p) = F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱}) - \sum_{j=۱}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{p} w(I_{i_j})$$

و

$$F(I_k) = F_{aL}(a_{i_k}) + F_{aR}(a_{i_k}) - \sum_{j=k}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=k+۱}^{p} w(I_{i_j})$$

برای یافتن اختلاف $F(I_p)$ و $F(I_k)$ تفاضل دو رابطه‌ی فوق را به دست می‌آوریم.

$$F(I_p) - F(I_k) = F_{aL}(a_{i_۱}) + F_{aR}(a_{i_۱}) - \sum_{j=۱}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{p} w(I_{i_j}) - [F_{aL}(a_{i_k}) +$$
$$F_{aR}(a_{i_k}) - \sum_{j=k}^{p-۱} w_{aR}(b_{i_j}) - \sum_{j=k+۱}^{p} w(I_{i_j})]$$

بنابراین

$$F(I_k) =$$
$$F(I_p) - F_{aL}(a_{i_۱}) - F_{aR}(a_{i_۱}) + F_{aL}(a_{i_k}) + F_{aR}(a_{i_k}) - \sum_{j=۱}^{k-۱} w_{aR}(b_{i_j}) - \sum_{j=۲}^{k} w(I_{i_j})]$$

□

۶.۵ الگوریتم

با استفاده از قضایا و نتایج فوق می‌توان الگوریتمی برای به دست آوردن هسته‌ی یک درخت بازه‌ای پیشنهاد کرد. به این صورت که اگر سمت چپ‌ترین بازه $I_۱$ باشد، $F(I_۱)$ را محاسبه می‌کنیم. طبق قضیه‌ی ۱.۶.۵ و نتیجه‌ی ۲.۵.۵ بازه‌های مینیمال نمی‌توانند کاندیدی برای قرارگرفتن در هسته باشند و بازه‌های ماکسیمال هسته را تشکیل می‌دهند. بنابراین برای ساختن هسته از $I_۱$ شروع کرده و از بین بازه‌هایی که شامل $b_۱$

هستند بازه‌ی ماکسیمال $I_۲$ را که بزرگترین نقطه‌ی انتهایی $b_۲$ را دارد انتخاب و آن را $RSUC(I_۱)$ نامیده و به مسیر $I_P = I_۱$ اضافه می‌کنیم. بنابراین $I_۲ = RSUC(I_۱)$ که در آن همانند [۴۷] برای هر نقطه q روی خط حقیقی $I_k = RSUC(q)$ اگر $b_k = max\{b_i \mid q \in I_i\}$، همچنین برای هر بازه I_j تعریف می‌کنیم :

$$RSUC(I_j) = RSUC(b_j)$$

بنابراین طبق نتیجه‌ی ۱.۵.۵ داریم:

$$F(I_p) = F(I_۱, I_۲) = F(I_۱) - w_{aR}(b) - w(I_۲)$$

حال $RSUC(I_۲)$ را یافته و آن را $I_۳$ نامیده و به مسیر اضافه می‌کنیم. با ادامه‌ی این روند رابطه‌ی بازگشتی زیر را به دست می‌آوریم.

$$F(I_p) = F(I_۱, I_۲, \ldots, I_k) = F(I_۱, I_۲, \ldots, I_{k-۱}) - w_{aR}(b_{k-۱}) - w(I_k)$$

اضافه شدن بازه‌ها به مسیر را تا سمت راست‌ترین بازه ادامه می‌دهیم. مسیر به دست آمده هسته است.

۱.۶.۵ الگوریتم

مباحث ذکر شده در بخش‌های فوق، الگوریتم زیر را تشکیل می‌دهد.

الگوریتم هسته‌ی یک درخت بازه‌ای

ورودی: یک درخت T .

خروجی: یک هسته‌ی S^* از T و $DISTSUM$ یا f_{best} آن.

فرض کنید تعداد بازه‌ها n است.

شروع

۱ـ قرار دهید $I_P = I_۱$ و $f_{best} := f(I_۱)$.

۲ـ قرار دهید $t := ۱$.

۳ـ **تا زمانی که** $t < n$ **مراحل زیر را انجام دهید.**

۳-۱ـ قرار دهید $I_{t+۱} := RSUC(I_t)$.

۳-۲ـ $I_{t+۱}$ را به I_p اضافه کنید.

۳-۳ـ $f_{best}(I_p) := f_{best}(I_p) - w_{aR}(b_t) - w(I_{t+۱})$

۳-۴ـ $t := t + ۱$

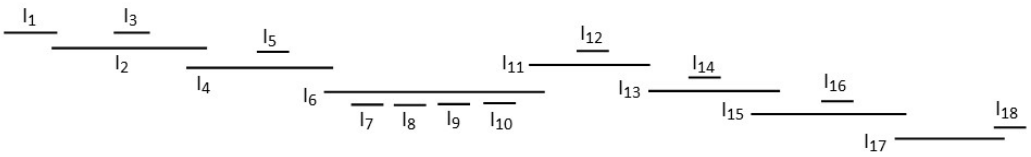
۴ـ مسیر I_p هسته‌ی درخت T است، بنابراین قراردهید $S^* := I_p$.

پایان

قضیه ۱.۶.۵. مسأله‌ی پیداکردن هسته‌ی یک درخت بازه‌ای که نقاط پایانی آن‌ها مرتب شده‌اند، می‌تواند در زمان $O(n)$ حل شود.

برهان. فرض کنیم $I = \{I_۱, I_۲, \ldots, I_n\}$ مجموعه‌ی بازه‌ها در درخت بازه‌ای G باشد. با توجه به الگوریتم ارائه شده، در بدترین حالت اگر n بازه‌ی ماکسیمال متصل به هم داشته باشیم، هسته شامل تمام بازه‌ها است. مقدار $F(I_n)$ می‌تواند در زمان ثابت محاسبه شود و چون n بازه داریم، مسأله‌ی پیداکردن هسته‌ی یک درخت بازه‌ای که نقاط پایانی آن‌ها مرتب شده‌اند، می‌تواند در زمان $o(n)$ حل شود. □

مثال ۱.۶.۵. بازه‌های شکل ۱۱.۵ را در نظر بگیرید. که وزن رئوس در جدول ۲.۵ داده شده است.



شکل ۱۱.۵: بازه‌های مثال ۱.۶.۵

$w_۹$	$w_۸$	$w_۷$	$w_۶$	$w_۵$	$w_۴$	$w_۳$	$w_۲$	$w_۱$
۰/۱	۰/۱	۰/۱	۱	۰/۱	۱	۰/۱	۱	۰/۱
$w_{۱۸}$	$w_{۱۷}$	$w_{۱۶}$	$w_{۱۵}$	$w_{۱۴}$	$w_{۱۳}$	$w_{۱۲}$	$w_{۱۱}$	$w_{۱۰}$
۰/۱	۱	۰/۱	۱	۰/۱	۱	۰/۱	۱	۰/۱

جدول ۲.۵: وزن‌های رئوس درخت شکل ۱۱.۵ در مثال ۱.۶.۵.

برای به دست آوردن هسته‌ی آن طبق الگوریتم از سمت چپ‌ترین بازه شروع می‌کنیم. بنابراین

$$F(I_p) = F(I_۱) = ۳۲/۷$$

سپس برای اضافه‌کردن بازه‌ی بعدی به مسیر، بازه‌ی $I_۲$ را انتخاب می‌کنیم زیرا $RSUC(I_۱) = I_۲$.

$$F(I_p) = F(I_۱, I_۲) = F(I_۱) - w_{aR}(b_۱) - w(I_۲) = ۳۲/۷ - ۷ - ۱ = ۲۴/۷$$

داریم:

$$RSUC(I_۲) = I_۴$$

بنابراین بازه‌ی $I_۴$ به مسیر اضافه می‌شود.

$$F(I_p) = F(I_۱, I_۲, I_۴) = F(I_۲) - w_{aR}(b_۲) - w(I_۴) = ۲۴/۷ - ۵/۹ - ۱ = ۱۷/۸$$

به همین ترتیب در هر مرحله $RSUC$ آخرین بازه را به دست آورده و به مسیر اضافه می‌کنیم. این روند را تا رسیدن به سمت راست‌ترین بازه یعنی $I_{۱۸}$ ادامه می‌دهیم.

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶) = F(I_۴) - w_{aR}(b_۴) - w(I_۶) = ۱۷/۸ - ۴/۸ - ۱ = ۱۲$$

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶, I_{۱۱}) = F(I_۶) - w_{aR}(b_۶) - w(I_{۱۱}) = ۱۲ - ۳/۴ - ۱ = ۷/۶$$

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶, I_{۱۱}, I_{۱۳}) = F(I_{۱۱}) - w_{aR}(b_{۱۱}) - w(I_{۱۳}) = ۷/۶ - ۲/۳ - ۱ = ۴/۳$$

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶, I_{۱۱}, I_{۱۳}, I_{۱۵}) = F(I_{۱۳}) - w_{aR}(b_{۱۳}) - w(I_{۱۵}) =$$
$$۴/۳ - ۱/۲ - ۱ = ۲/۱$$

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶, I_{۱۱}, I_{۱۳}, I_{۱۵}, I_{۱۷}) = F(I_{۱۵}) - w_{aR}(b_{۱۵}) - w(I_{۱۷}) =$$
$$۲/۱ - ۰/۱ - ۱ = ۱$$

$$F(I_p) = F(I_۱, I_۲, I_۴, I_۶, I_{۱۱}, I_{۱۳}, I_{۱۵}, I_{۱۷}, I_{۱۸}) = F(I_{۱۷}) - w_{aR}(b_{۱۷}) - w(I_{۱۸}) =$$
$$۱ - ۰/۱ = ۰/۹$$

بنابراین هسته‌ی گراف بازه‌ای فوق مسیر $p = \{I_۱, I_۲, I_۴, I_۶, I_{۱۱}, I_{۱۳}, I_{۱۵}, I_{۱۷}, I_{۱۸}\}$ است و $F(I_p) = ۰/۹$ می‌باشد.

۷.۵ نتیجه گیری

در این فصل به بررسی مساله‌ی پیداکردن هسته روی درخت‌های بازه‌ای پرداختیم و نشان دادیم بازه‌هایی که روی هسته‌ی یک درخت قرار دارند نمی‌توانند بازه‌ای غیر ماکسیمال باشند. سپس الگوریتمی با پیچیدگی زمانی $O(n)$ برای پیداکردن هسته‌ی یک درخت بازه‌ای ارائه دادیم. در مطالعات بعدی می‌توان هسته‌ی درخت را با محدودیت طول در نظرگرفت. بنابراین هدف یافتن هسته‌ای با طول L روی درخت بازه‌ای است. همچنین می‌توان مساله را از یافتن هسته به $(k,l)-$هسته تغییر داد که در این حالت هدف یافتن زیردرختی با طول حداکثر L و تعداد برگ‌های حداکثر K است که مجموع فاصله‌های تمام بازه‌ها تا این زیردرخت کمینه شود.

مراجع

[۱] امینی ف.، (۱۳۸۸)، ”گراف‌های بازه‌ای کاوشگر“، **مجله تحقیق در عملیات و کاربردهای آن**، شماره ۳، دوره ۲۲ ص ۶۱-۷۰.

[۲] قاسمی ح.، (۱۳۹۱)، پایان‌نامه کارشناسی‌ارشد: ”مسأله مکانیابی p-ماکسین“، دانشگاه صنعتی شاهرود، دانشکده ریاضی.

[۳] ربیعی ا.، (۱۳۹۴)، پایان‌نامه کارشناسی‌ارشد: ”گراف بازه‌ای و کاربردهایش“، دانشگاه صنعتی شاهرود، دانشکده ریاضی.

[4] Arikati S.R. and Pandu Rangan C. (1990), ”Linear algorithm for optimal path cover problem on interval graphs”, **Inf. Process. Lett.**, 35, pp. 149-153.

[5] Balinski M.L. (1965), ”Integer programming: Methods, uses, computation”, **Management Sci**, 12, pp. 253-313.

[6] Beasley J.E. (1990), ”OR-Library: distributing test problems by electronic mail”, **Journal of the Operational Research Society**, 41, pp. 1069-1072.

[7] Becker R. I. (1990), ”Inductive algorithms on finite trees”, **Quaest Math.**, 13 , pp. 165-181.

[8] Becker R. I., Lari I., Storchi G. and Scozzari A. (2002), ”Efficient algorithms for finding the (k, l)-core of tree networks”, **Networks**, 40 pp. 208-215.

[9] Benkoczia R., Bhattacharya B.K. and Breton D. (2006), ”Efficient computation of 2-medians in a tree network with positive/negative weights”, **Discrete Mathematics**, 306, pp. 1505-1516.

[10] Berge C. (1973), ”**Graphs and Hypergraphs**”, Vol. 6, University Paris, pp. 372.

[11] Bespamyatnikh S., Bhattacharya B., Keil M., Kirkpatrick D. and Segal M. (2002), ”Efficient algorithms for centers and medians in interval and circular-arc graphs”, **Networks**, 29, pp. 144-152.

[12] Burkard R.E. and Krarup J. (1998), ”A linear algorithm for the pos/neg-weighted 1-median problem on a cactus”, **Computing**, 60, pp. 193-215.

[13] Burkard R.E., Çela E. and Dollani H. (2000), ”2-Median in trees with pos/neg weights”, **Discrete Appl. Math.**, 105, pp. 51-71.

[14] Burkard R.E. and Fathali J. (2007), ”A polnomial method for the pos/neg weighted 3-median problem on a tree”, **Mathematical Methods of Operations Research**, 65, pp. 229-238.

[15] Burkard R.E. , Fathali J. and Kakhki H.T. (2007), ”The p-maxian problem on a tree”, **Operations Research Letters**, 35, pp. 331-335.

[16] Chen D. , Lee D.T. , Sridhar R. and Sekharam C. (1998), ”Solving the all-pair shortest path query on interval and circular-arc graphs”, **Networks**, 31, pp. 249-258.

[17] Cheng T. C. E. , Kang L. Y. , Ng C. T. (2007), ”An improved algorithm for the p-center problem on interval graphs with unit lengths”, **Comput. Oper. Res.**, 34, pp. 2215-2222.

[18] Cheng Y. and Kang L. (2010), ”The p-maxian problem on interval graphs”, **Discrete Applied Mathematics**, 158, pp. 1986-1993.

[19] Church R. L. and Revelle C. S. (1976), ”Theoretical and comptational links between p-median location set-covering and maximal covering location problem”, **Geographical Analysis**, 8, pp. 406-415.

[20] Church R.L. and Garfinkel R.S. (1978), ”Locating an obnoxious facility on a network”, **Transportation Science**, 12, pp. 107-118.

[21] Drezner H. and Hamacher H. (2002), ”Facility Location: Application and Theory ”, **Springer-Verlag**, Berlin.

[22] Daskin M. S. (1995), ” Network and Discrete Location: Models Algorithms and Applications”, Wiley, New York.

[23] Eiselt H. A. and Laporte G. (1929), ”Objectives in location problems”, **Facility Location**, 8 pp. 41–57.

[24] Farber M. (1984), ”Domination, independent domination, and duality in strongly chordal graphs”, **Discrete Applied Mathematics**, 7, pp. 115-130.

[25] Francis R. L. , McGinnis L. F. and White J. A. (1992), ”Facility layout and location: Analytical approach”, 2nd ed, Prentice Hall.

[26] Garey M. R. and johnson D. S. (1979), ”Computers and Intractability: A Guide to the Theory of NP-Completeness”, Freeman, New York.

[27] Gavish B. and Sridhar S. (1995), ”Computing the 2-median on tree network in o(nlogn) time”, **Networks**, 26, pp. 305-317.

[28] Gavril F. (1972), ”Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of a chordal graph”, **SIAM Journal on Computing**, 1, pp. 180-187.

[29] Goldberg D.E. (1989), ”Genetic Algorithms In Search, Optimization And Machine Learning”, Menlo Park, CA: Addison-Weseley.

[30] Goldman A.J. (1971), ”Optimum center location in simple networks”, **Transport Sci**, 5 pp. 212–221 .

[31] Golumbic M.C. (1980), ”Algorithmic Graph Theory and Perfect Graphs”, Academic Press, New York.

[32] Hakimi S. L. (1964), ”Optimum locations of switching centers and the absolute centers and medians of a graph”, **Operations research**, 12 pp. 450–459.

[33] Hakimi S. L., Schmeichel E.F. and Labbe M. (1993), ”On locating path or tree shaped facilities on network”, **Networks**, 23 pp. 543–555.

[34] Hale T. (2004), ”Trevor Hales location science references”, http:/wwwent.ohiou.edu/thale/thlocation.html.

[35] Halldorsson M. M., Kortsarz G. and Shachnai H. (2003), ”Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs”, **Algorithmica**, 37, pp. 187-209.

[36] Hamacher H. W. and NickelS. (1998), ”Classification of location models”, **Location Science**, 6 pp. 229–242.

[37] Handler G. Y. and Mirchandani P. B. (1979), ”Location on network; Theory and Algorithms”, MIT Press, Cambridge.

[38] Hansen P., Labbe M. , Peeters D. and Thisse J. F. (1987), ”Single facility Location on network”, **Annals of Dscrete Math**, 31 pp. 113–146.

[39] Hotelling H. (1929), ”Stability in competetion”, **Economic Journal**, 39 pp. 41–57.

[40] Ioannidou K. , Mertzios G. B. and Nikolopoulos S. D. (2011), ”The Longest Path Problem has a Polynomial Solution on Interval Graphs”, **Algorithmica**, 61, pp. 320-341.

[41] Irani S. and Leung V. (2003), ”Scheduling with conflicts on bipartite interval graphs”, **J. Scheduling**, 6, pp. 287-307.

[42] Krarup J. and Pruzan P. M. (1983), ”The simple plant location problem: survey and synthesis”, **European Journal of Operational Research**, 12 pp. 36–81.

[43] Kuehn A. A. and Hamburger M. J. (1963), ”A heuristic program for locating warehouses”, **Management Sci**, 9 pp. 643–666.

[44] Manne A. (1964), ”Plant location under economics of scale-decentralization and computation”, **Management Sci**, 11 pp. 213-235.

[45] Minieka E. and Patel N.H. (1983), ”On finding the core of a tree with a specified length”, **J. Alg.**, 4, pp. 345-352.

[46] Mirchandani P. B. and Francis R. (2004), ”Discrete Location Theory”, **Fuzzy sets and systems**, 142 pp. 393–405.

[47] Morgan C. A. and Slater P. J. (1980), ”A linear algorithm for a core of a tree”, **Journal of Algorithms**, 1, pp. 247-258.

[48] Motevalli S., Fathali J. and Zaferanieh M. (2015), ”An efficient algorithm for finding the semi-obnoxious (k, l)-core of a tree”, **Journal of Mathematical Modeling**, 3 pp. 129-144.

[49] Owen S.H. and Daskin M. S. (1998), ”Strategic facility location: A review”, **European Journal of Operation Research**, 111 pp. 423–447.

[50] Peng S., Stephens A.B., and Yesha Y. (1993), ”Algorithms for a core and a k-tree core of a tree”, **J. Alg.**, 15, pp. 143-159.

[51] Peng S. and Lo W. (1996), ”Efficient algorithms for finding a core of a tree with a specified length”, **Journal of Algorithms**, 20, pp. 445-458.

[52] Rahbari M. and Fathali J. (2015), ”A hybrid algorithm for the path center problem”, **Global Analysis and Discrete Mathematics**, 1, pp. 83-92.

[53] Reeves C.R. (1993), ”Genetic Algorithms. In Modern Heuristic Techniques for Combinatorial Problems”,(C.R Reeves, Eds.), 4.

[54] Revelle C.S. and Swain R.W. (1970), ”Central facility location”, **Geographical Analysis**, 1, pp. 30-42.

[55] Scaparra M. P. and Scutella M. G. (2001), ”Strategic facility location customers: Building blocks of location models: A survey”, Thecnical Report: tr-01-18, University of Piza, Italy.

[56] Shioura A. and Uno T. (1997), ”A linear time algorithm for finding a k-tree core”, **J. Alg.**, 23, pp. 281-290.

[57] Sivanandam S.N. and Deepa S.N. (2008), ”Introduction to Genetic Algorithms”, Springer-Verlag Berlin Heidelberg.

[58] Tamir A. (1991), ”Obnoxious facility location on graphs”, **SIAM Journal on Discrete Mathematics**, 4, pp. 550-567.

[59] Teitz M. B. and Bart P. (1968), ”Heuristic methods for estimating the generalized vertex median of a weghted graph”, **Operations Research**, 16, pp. 955-961.

[60] Ting S.S. (1984), ”A linear-time algorithm for maxisum facility location on tree networks”, **Transportation Science**, 18, pp. 76-84.

[61] Uehara R. and Uno Y. (2004), ”Efficient algorithms for the longest path problem”, **Algorithms and Computation**, 3341, pp. 871-883.

[62] Zaferanieh M. and Fathali J. (2009), ”Ant colony and simulated annealing algorithms for finding the core of a graph”, **World Applied Sciences Journal**, 7, pp. 1335-1341.

[63] Zaferanieh M. and Fathali J. (2012), ”Finding a core of a tree with pos/neg weight”, **Math Meth Oper Res.**, 26, pp. 147-160.

[64] Zelinka B. (1968), ”Medians and peripherians of trees”, **Archvum Mathematicum**, 4, pp. 87-95.

پیوست آ

برنامه‌های کامپیوتری

آ.۱ کد برنامه متلب مربوط به الگوریتم ژنتیک

```
function main
tic
clc
clear all
close all
np=45;% tedad jameat avalie
fid = fopen('pmed35.txt', 'r');
fidd = fopen('f22wpmmut35.txt', 'w');
n=fscanf(fid, '%f',1)
Edges=fscanf(fid, '%f',1)
pmed=fscanf(fid, '%f',1)
K=0.1*n
k1=K;
L=(n*20)/10
itration=1
```

```
n10=n/10;
for i=1:n10
w(i)=-1;
end
n10=n10+1;
for i=n10:n
w(i)=1;
end
w;
for q=1:Edges
i=fscanf(fid, ’%f’,1);
j=fscanf(fid, ’%f’,1);
adjacent(i,j)=fscanf(fid, ’%f’,1);
end
for i=1:n
for j=1:n
if (adjacent(i,j)==0  i =j)
adjacent(i,j)=inf;
end
end
end
adjacent;
distance=floyd(adjacent);
for i=1:n
wa(i)=i;
end
    wsort,wsorta
=sorting(w,wa);
leafs=leaf(adjacent);
s=zeros(n,n,np);
fitnes=zeros(1,np);
s=initial;
q=0;
for q=1:np
fitnes(q)=fitness1(s(:,:,q));
```

```
end
fitnes
fprintf(fidd, 'The initial population=');
fprintf(fidd, '%6.1f ', fitnes);
minf=min(fitnes)
it=1;
while (itration<=100  it<=20)
parent1=zeros(n,n);
parent2=zeros(n,n);
fparent1=0;
fparent2=0;
    parent1,fparent1
=bestfit(fitnes);
nparent2=randi(np);
fparent2=fitnes(nparent2);
parent2=s(:,:,nparent2);
child1=zeros(n,n);
child2=zeros(n,n);
    child1,child2
=crossover(parent1,parent2);
fitnesch1=fitness1(child1);
fitnesch2=fitness1(child2);
s=replace(child1,fitnesch1,s,fitnes);
s=replace(child2,fitnesch2,s,fitnes);
q=0;
for q=1:np
fitnes(q)=fitness1(s(:,:,q));
end
fitnes;
s;
nchildm=randi(np);
fchildm=fitnes(nchildm);
childm=s(:,:,nchildm);
childmnew=motation(childm);
fchildmnew=fitness1(childmnew);
```

```
s=replace(childmnew,fchildmnew,s,fitnes);
q=0;
for q=1:np
fitnes(q)=fitness1(s(:,:,q));
end
minfit=min(fitnes);
if minfit==minf
it=it+1;
elseif minfit<minf
minf=minfit;
it=1;
end
i=itration
minfit
fprintf(fidd,'%d ', i);
fprintf(fidd, '%6.1f ', minfit);
itration=itration+1;
end
fclose(fid);
fclose(fidd);
toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [distance]=floyd(adjacent)
m=size(adjacent);
m=m(1);
distance=adjacent;
for i1=1:m
row=distance(i1,:);
column=distance(:,i1);
for j1=1:m
for k=1:m
min=row(j1)+column(k);
if min<distance(j1,k);
distance(j1,k)=min;
```

```
end
end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [disch,child]=lcut(disch,child)
for i1=1:n
for j1=1:n
if (disch(i1,j1) > L  disch(i1,j1) =inf)
child(i1,j1)=0;
disch(i1,j1)=disch(i1,j1)-disch(i1,j1);
end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [wsort,wsorta]=sorting(w,wa)
wsort=w;
temp=0;
for i1=1:n
wsorta(i1)=wa(i1);
end
for i1=1:n
for j1=i1+1:n
if wsort(j1)>wsort(i1);
temp=wsort(i1);
wsort(i1)=wsort(j1);
wsort(j1)=temp;
temp=wsorta(i1);
wsorta(i1)=wsorta(j1);
wsorta(j1)=temp;
end
end
```

```
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [leafs,nleaf]=leaf(tree)
leafs=zeros(1,n);
nleaf=0;
t=0;
for i1=1:n
if tree(i1,i1) =0
t=i1;
end
end
if t =0
leafs(1)=t;
nleaf=1;
else
for i1=1:n
t=0;
for j1=1:n
if (i1 =j1  tree(i1,j1)==1);
t=t+1;
end
end
if t==1
nleaf=nleaf+1;
leafs(nleaf)=i1;
end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [s]=initial
K=K-2;
for i1=1:np
```

```
l=0;
k=1;
subtree=zeros(n,n);
bestsubtree=zeros(n,n);
fitnesin=0;
p=zeros(1,n+1);
l1=zeros(1,n);
wadj1=zeros(1,n);
wadj2=zeros(1,n);
p(1)=wsorta(i1);
endp=wsorta(i1);
q=1;
f=0;
t=0;
tt=0;
for j1=1:n
f=f+w(j1)*distance(j1,wsorta(i1));
end
fitnes(i1)=f;
while (l<L  p(q) =0)
t=1;
%finding adjacents of endp%
adj1=zeros(1,n);
for j1=1:n
f=1;
h=0;
if (endp =j1  adjacent(endp,j1) =inf);
while p(f) =0
if p(f)==j1
h=1;
end
f=f+1;
end
if h==0
```

```
adj1(t)=j1;
wadj1(t)=w(j1);
t=t+1;
end
end
end
if adj1(1) =0
%finding max weight in adjacents of endp%
max=wadj1(1);
u=adj1(1);
f=1;
while adj1(f) =0
if wadj1(f)>max ;
max=wadj1(f);
u=adj1(f);
end
f=f+1;
end
l=l+adjacent(endp,u);
q=q+1;
p(q)=u;
endp=u;
subtree(p(q-1),p(q))=1;
subtree(p(q),p(q-1))=1;
fitnesin=fitness1(subtree);
if fitnesin<fitnes(i1)
fitnes(i1)=fitnesin;
bestsubtree=subtree;
tt=1;
end
else p(q)=0;
end
end
if tt==1
```

```
%internal vertexes of p% interv=zeros(1,n);
for j1=2:q-1
interv(j1-1)=p(j1);
end
%finding adjacents of internal vertexes of p%
k=1;
while k<=K
t=1;
adj2=zeros(1,n);
for j1=1:q-2
for e=1:n
f=1;
h=0;
z=interv(j1);
if (z =e  adjacent(z,e) =inf);
while p(f) =0
if p(f)==e
h=1;
end
f=f+1;
end
if h==0
adj2(t)=e;
wadj2(t)=w(e);
t=t+1;
end
end
end
end
%finding max weight in adjacents of internal vertexes of p%
if adj2(1) =0
max=wadj2(1);
u=adj2(1);
f=1;
```

```
while (adj2(f) =0  f<n)
if wadj2(f)>max ;
max=wadj2(f);
u=adj2(f);
end
f=f+1;
end
else u=0;
end
%l=l+adjacent(endp,u);
l1(k)=u;
if u =0
for j1=1:q-2
if(interv(j1) =u  adjacent(u,interv(j1)) =inf);
subtree(interv(j1),u)=1;
subtree(u,interv(j1))=1;
fitnesin=fitness1(subtree);
if fitnesin<fitnes(i1)
fitnes(i1)=fitnesin;
bestsubtree=subtree;
t=1;
end
end
end
end
k=k+1;
end
else
bestsubtree(wsorta(i1),wsorta(i1))= fitnes(i1);
end
%deleting value for majer dimation
%for bestsubtree
v=0;
for i5=1:n
```

```
if bestsubtree(i5,i5) =0
v=i5;
end
end
if v =0
i5=1;
t=0;
while (i5<=n  t==0)
for j5=i5+1:n
if bestsubtree(i5,j5) =0
t=1;
end
end
i5=i5+1;
end
if t==1
bestsubtree(v,v)=0;
end
end
s(:,:,i1)=bestsubtree;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [parent1,fparent1]=bestfit(fitnes)
fparent1=fitnes(1);
parent1=s(:,:,1);
i1=1;
for i1=2:np
if fitnes(i1)<fparent1
fparent1=fitnes(i1);
parent1=s(:,:,i1);
end
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [child1,child2]=crossover(parent1,parent2)
child1=zeros(n,n);
child2=zeros(n,n);
f1=0;
f2=0;
h=0;
m=0;
%finding alone vertices
for i5=1:n
if parent1(i5,i5) =0
f1=i5;
end
end
for i5=1:n
if parent2(i5,i5) =0
f2=i5;
end
end
%both of parents are a vertex
if ((f1 =0)(f2 =0))
child1(f1,f1)=parent1(f1,f1);
child1(f2,f2)=parent2(f2,f2);
child2(f1,f1)=parent1(f1,f1);
child2(f2,f2)=parent2(f2,f2);
else
%finding common vertices
for i5=1:n
for j5=1:n
if parent1(i5,j5)==parent2(i5,j5)
child1(i5,j5)=parent1(i5,j5);
child2(i5,j5)=parent1(i5,j5);
end
if ((parent1(i5,j5)==1)(f2==i5))
```

```
child1(i5,i5)=parent2(i5,i5);
child2(i5,i5)=parent2(i5,i5);
end
if ((parent2(i5,j5)==1)(f1==i5))
child1(i5,i5)=parent1(i5,i5);
child2(i5,i5)=parent1(i5,i5);
end
end
end
end
%fulling vacant spaces
b=0;
for i5=1:n
h=0;
h1=0;
h2=0;
p1=zeros(1,n);
p2=zeros(1,n);
for j5=1:n
if ((child1(i5,j5)==0  child2(i5,j5)==0)(parent1(i5,j5)==1 | parent2(i5,j5)==1))
h=1;
end
if parent1(i5,j5)==1
h1=h1+1;
p1(h1)=j5;
end
if parent2(i5,j5)==1
h2=h2+1;
p2(h2)=j5;
end
end
if (h==1)  (h1 =0 | h2 =0)
f1=0;
f2=0;
```

```
%select from parent1 for child1
if mod(b,3)==0
while (h1>0  f1==0)
p3=zeros(1,n);
i1=i5;
j5=p1(h1);
p3(1)=i5;
p3(2)=p1(h1);
m=2;
f2=0;
t2=j5;
while (f2==0  m<n)
i1=t2;
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==0  t2 =0)
m=m+1;
p3(m)=t2;
else
f2=1;
end
end
i1=p3(m);
t1=0;
```

```
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==1 | t2==0)
f1=1;
end
h1=h1-1;
end
i1=i5;
for j5=1:m
child1(i1,p3(j5))=parent1(i1,p3(j5));
child1(p3(j5),i1)=parent1(i1,p3(j5));
i1=p3(j5);
end
%select from parent2 for child1
else
while (h2>0  f1==0)
p3=zeros(1,n);
i1=i5;
j5=p2(h2);
p3(1)=i5;
p3(2)=p2(h2);
m=2;
f2=0;
t2=j5;
while (f2==0  m<n)
```

```
i1=t2;
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent2(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==0  t2 =0)
m=m+1;
p3(m)=t2;
else
f2=1;
end
end
i1=p3(m);
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==1 | t2==0)
```

```
f1=1;
end
h2=h2-1;
end
i1=i5;
for j5=1:m
child1(i1,p3(j5))=parent2(i1,p3(j5));
child1(p3(j5),i1)=parent2(i1,p3(j5));
i1=p3(j5);
end
end
%select from parent1 for child2
f1=0;
f2=0;
if mod(b,3) =0
while (h1>0  f1==0)
p3=zeros(1,n);
i1=i5;
j5=p1(h1);
p3(1)=i5;
p3(2)=p1(h1);
m=2;
f2=0;
t2=j5;
while (f2==0  m<n)
i1=t2;
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child2(i1,j5) =0
t1=1;
end
```

```
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==0  t2 =0)
m=m+1;
p3(m)=t2;
else
f2=1;
end
end
i1=p3(m);
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==1 | t2==0)
f1=1;
end
h1=h1-1;
end
i1=i5;
for j5=1:m
child2(i1,p3(j5))=parent1(i1,p3(j5));
child2(p3(j5),i1)=parent1(i1,p3(j5));
i1=p3(j5);
```

```
end
%select from parent2 for child2
else
while (h2>0  f1==0)
p3=zeros(1,n);
i1=i5;
j5=p2(h2);
p3(1)=i5;
p3(2)=p2(h2);
m=2;
f2=0;
t2=j5;
while (f2==0  m<n)
i1=t2;
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child2(i1,j5) =0
t1=1;
end
if (parent2(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==0  t2 =0)
m=m+1;
p3(m)=t2;
else
f2=1;
end
end
i1=p3(m);
```

```
t1=0;
t2=0;
j5=0;
while (t1==0  j5<n)
j5=j5+1;
if child1(i1,j5) =0
t1=1;
end
if (parent1(i1,j5) =0  j5 =p3(m-1))
t2=j5;
end
end
if (t1==1 | t2==0)
f1=1;
end
h2=h2-1;
end
i1=i5;
for j5=1:m
child2(i1,p3(j5))=parent2(i1,p3(j5));
child2(p3(j5),i1)=parent2(i1,p3(j5));
i1=p3(j5);
end
end
b=b+1;
end
end
%cuting pathes with length biger than L
disch1=zeros(n,n);
disch2=zeros(n,n);
for i1=1:n
for j1=i1:n
if (child1(i1,j1) =0  i1 =j1)
adchild1(i1,j1)=adjacent(i1,j1);
```

```
adchild1(j1,i1)=adjacent(i1,j1);
elseif (child1(i1,j1)==0  i1 =j1)
adchild1(i1,j1)=inf;
adchild1(j1,i1)=inf;
elseif (child1(i1,j1) =0  i1==j1)
adchild1(i1,j1)=0;
adchild1(j1,i1)=0;
end
if (child2(i1,j1) =0  i1 =j1)
adchild2(i1,j1)=adjacent(i1,j1);
adchild2(j1,i1)=adjacent(i1,j1);
elseif (child2(i1,j1)==0  i1 =j1)
adchild2(i1,j1)=inf;
adchild2(j1,i1)=inf;
elseif (child2(i1,j1) =0  i1==j1)
adchild2(i1,j1)=0;
adchild2(j1,i1)=0;
end
end
end
disch1=floyd(adchild1);
disch2=floyd(adchild2);
%child1
t=0;
for i1=1:n
for j1=i1:n
if (disch1(i1,j1) > L  disch1(i1,j1) =inf)
t=1;
end
end
end
if t==1
n1=1;
while (t =0  n1<n)
```

```
t=0;
    disch1,child1
=lcut(disch1,child1);
for i1=1:n
for j1=i1:n
if (disch1(i1,j1) > L  disch1(i1,j1) =inf)
t=1;
end
end
end
n1=n1+1;
end
end
%child2
t=0;
for i1=1:n
for j1=i1:n
if (disch1(i1,j1) > L  disch1(i1,j1) =inf)
t=1;
end
end
end
if t==1
n1=1;
while (t =0  n1<n)
t=0;
    disch1,child1
=lcut(disch1,child1);
for i1=1:n
for j1=i1:n
if (disch1(i1,j1) > L  disch1(i1,j1) =inf)
t=1;
end
end
end
end
```

```
n1=n1+1;
end
end
%cuting leafs biger than k
%child1
leafs1=zeros(1,n);
nleaf1=0;
    leafs1,nleaf1
=leaf(child1);
wleaf1=zeros(1,n);
for i1=1:nleaf1
wleaf1(i1)=w(leafs1(i1));
end
wleaf1;
b1=nleaf1;
d=b1;
K=k1;
t=1;
while (b1>K  t<=n)
    lsort,wlsort
=sorting(leafs1,wleaf1);
i1=1;
t=0;
j1=lsort(1);
while (i1<=n  t==0)
if child1(i1,j1) =0
child1(i1,j1)=0;
child1(j1,i1)=0;
t=1;
end
i1=i1+1;
end
if d==2
child1(lsort(2),lsort(2))=0;
end
    leafs1,nleaf1
```

```
=leaf(child1);
b1=nleaf1;
t=t+1;
end
%child2
leafs2=zeros(1,n);
nleaf2=0;
    leafs2,nleaf2
=leaf(child2);
wleaf2=zeros(1,n);
for i1=1:nleaf2
wleaf2(i1)=w(leafs2(i1));
end
wleaf2;
b1=nleaf2;
d=b1;
K=k1;
t=1;
while (b1>K  t<=n)
    lsort,wlsort
=sorting(leafs2,wleaf2);
i1=1;
t=0;
j1=lsort(1);
while (i1<=n  t==0)
if child2(i1,j1) =0
child2(i1,j1)=0;
child2(j1,i1)=0;
t=1;
end
i1=i1+1;
end
if d==2
child2(lsort(2),lsort(2))=0;
end
    leafs2,nleaf2
```

```
=leaf(child2);
b1=nleaf2;
t=t+1;
end
%deleting value for majer dimation
%for child1
v=0;
for i1=1:n
if child1(i1,i1) =0
v=i1;
end
end
if v =0
i1=1;
t=0;
while (i1<=n  t==0)
for j1=i1+1:n
if child1(i1,j1) =0
t=1;
end
end
i1=i1+1;
end
if t==1
child1(v,v)=0;
end
end
%for child2
v=0;
for i1=1:n
if child2(i1,i1) =0
v=i1;
end
end
```

```
if v =0
i1=1;
t=0;
while (i1<=n  t==0)
for j1=i1+1:n
if child2(i1,j1) =0
t=1;
end
end
i1=i1+1;
end
if t==1
child2(v,v)=0;
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fitnesch]=fitness1(child)
s1=zeros(1,n);
t1=zeros(1,n);
f1=1;
f2=1;
t=0;
for i1=1:n
if child(i1,i1) =0
fitnesch=child(i1,i1);
t=1;
end
end
if t==0
for i1=1:n
h=0;
for j1=1:n
if (child(i1,j1)==1)
```

```
h=1;
end
end
if h==0
s1(f1)=i1;
f1=f1+1;
else
t1(f2)=i1;
f2=f2+1;
end
end
minpath=zeros(1,n);
for i1=1:f1-1
minpath(i1)=inf;
for j1=1:f2-1
if distance(s1(i1),t1(j1))<minpath(i1)
minpath(i1)=distance(s1(i1),t1(j1));
end
end
end
wminpath=zeros(1,n);
for i1=1:f1-1
wminpath(i1)=minpath(i1)*w(s1(i1));
end
fitnesch=sum(wminpath);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [s]=replace(child,fitnesch,s,fitnes)
t=fitnes(1);
ta=1;
for i1=1:np
if fitnes(i1)>t
t=fitnes(i1);
```

```
ta=i1;
end
end
if fitnes(ta)>fitnesch
fitnes(ta)=fitnesch;
for i1=1:n
for j1=1:n
s(i1,j1,ta)=child(i1,j1);
end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [childmnew]=motation(childm)
for i1=1:n
for j1=1:n
childmnew(i1,j1)=childm(i1,j1);
end
end
t=0;
for i1=1:n
if childmnew(i1,i1) =0
t=i1;
end
end
childmnew;
if t =0
v=t;
adjacents=zeros(1,n);
adjacents=findadj(childmnew,v,adjacent);
t=1;
while adjacents(t) =0
t=t+1;
end
```

```
nadj=t-1;
if nadj>0
t=1;
wadjacents=zeros(1,n);
for i1=nadj+1:n
wadjacents(i1)=-inf;
end
while adjacents(t) =0
wadjacents(t)=w(adjacents(t));
t=t+1;
end
    wsortadj,awsortadj
=sorting(wadjacents,adjacents);
childmnew(v,v)=0;
for i1=1:nadj
if wsortadj(i1)>0
childmnew(awsortadj(i1),v)=1;
childmnew(v,awsortadj(i1))=1;
end
end
end
else
    leafchm,nleafchm
=leaf(childmnew);
for i1=1:nleafchm
f1=0;
j1=1;
while (j1<=n  f1==0)
if childmnew(leafchm(i1),j1)==1
i2=j1;
f2=0;
j2=1;
while (j2<=n  f2==0)
if (adjacent(i2,j2) =inf  i2 =j2  leafchm(i1) =j2)
f3=0;
```

```
for i3=1:nleafchm
if j2==leafchm(i3)
f3=1;
end
end
if f3==0
childmnew(i2,j2)=1;
childmnew(leafchm(i1),j1)=0;
childmnew(j2,i2)=1;
childmnew(j1,leafchm(i1))=0;
f2=1;
end
end
j2=j2+1;
end
f1=1;
end
j1=j1+1;
end
end
childmnew;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%cuting pathes with length biger than L
disch=zeros(n,n);
for i1=1:n
for j1=i1:n
if (childmnew(i1,j1) =0  i1 =j1)
adchild(i1,j1)=adjacent(i1,j1);
adchild(j1,i1)=adjacent(i1,j1);
elseif (childmnew(i1,j1)==0  i1 =j1)
adchild(i1,j1)=inf;
adchild(j1,i1)=inf;
elseif (childmnew(i1,j1) =0  i1==j1)
```

```
adchild(i1,j1)=0;
adchild(j1,i1)=0;
end
end
end
disch=floyd(adchild);
    leafchm,nleafchm
=leaf(childmnew);
%nleafchm=floor(nleafchm/2);
for i1=1:nleafchm
f=0;
for j1=1:n
if (disch(j1,i1) =inf  disch(j1,i1)>L)
f=1;
end
end
if f==0
adjacents=zeros(1,n);
adjacents=findadj(childmnew,i1,adjacent);
t=1;
while adjacents(t) =0
t=t+1;
end
nadj=t-1;
if nadj>0
t=1;
wadjacents=zeros(1,n);
for i2=nadj+1:n
wadjacents(i2)=-inf;
end
while adjacents(t) =0
wadjacents(t)=w(adjacents(t));
t=t+1;
end
    wsortadj,awsortadj
```

```
=sorting(wadjacents,adjacents);
%nadj=nadj-(nadj/2);
for i2=1:nadj
if wsortadj(i2)>0
childmnew(awsortadj(i2),i1)=1;
childmnew(i1,awsortadj(i2))=1;
end
end
end
end
end
%cuting leafs biger than k
leafs1=zeros(1,n);
nleaf1=0;
    leafs1,nleaf1
=leaf(childmnew);
wleaf1=zeros(1,n);
for i1=1:nleaf1
wleaf1(i1)=w(leafs1(i1));
end
wleaf1;
b1=nleaf1;
d=b1;
K=k1;
while b1>K
    lsort,wlsort
=sorting(leafs1,wleaf1);
i1=1;
t=0;
j1=lsort(1);
while (i1<=n  t==0)
if childmnew(i1,j1) =0
childmnew(i1,j1)=0;
childmnew(j1,i1)=0;
t=1;
```

```
end
i1=i1+1;
end
if d==2
childmnew(lsort(2),lsort(2))=0;
end leafs1,nleaf1
=leaf(childmnew);
b1=nleaf1;
end
%%%%%%%%%%%%
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [adjes]=findadj(a,v,adjacent)
adjes=zeros(1,n);
t=1;
for j1=1:n
if (v =j1  adjacent(v,j1) =inf);
adjes(t)=j1;
t=t+1;
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

Aabstract

In this thesis the (k,l)-core problem on a network is considered. A polynomial algorithm for solving this problem on a tree with positive and negative weight is presented. This problem on the general network is NP-hard. So for finding the (k,l)-core of a network, three genetic algorithms are designed. Also, computational results of these algorithms are compared together in terms of speed and efficiency. Then the problem of finding the 2-(k,l)-core of a tree is investigated and an algorithm that is a generalization of the (k,l)-core of a tree algorithm is introduced for solving 2-(k,l)-core problem. In addition, the problem of finding core of a network is considered on the weighted interval graphs and a new method is presented for solving this problem.
Keywords: Core; Facility Location; (k,l)-core; Genetic Algorithm; 2-(k,l)-Core; Interval Trees.

Shahrood University of Technology

Faculty Of Mathematical Sciences

PhD Thesis in: Operation Research

Investigation of the (k,l)-Core Problem on the Network

By: Samane Motevalli Ashkezari

Supervisor

Jafar Fathali

Advisor

Mahdi Zaferanieh

November 2018