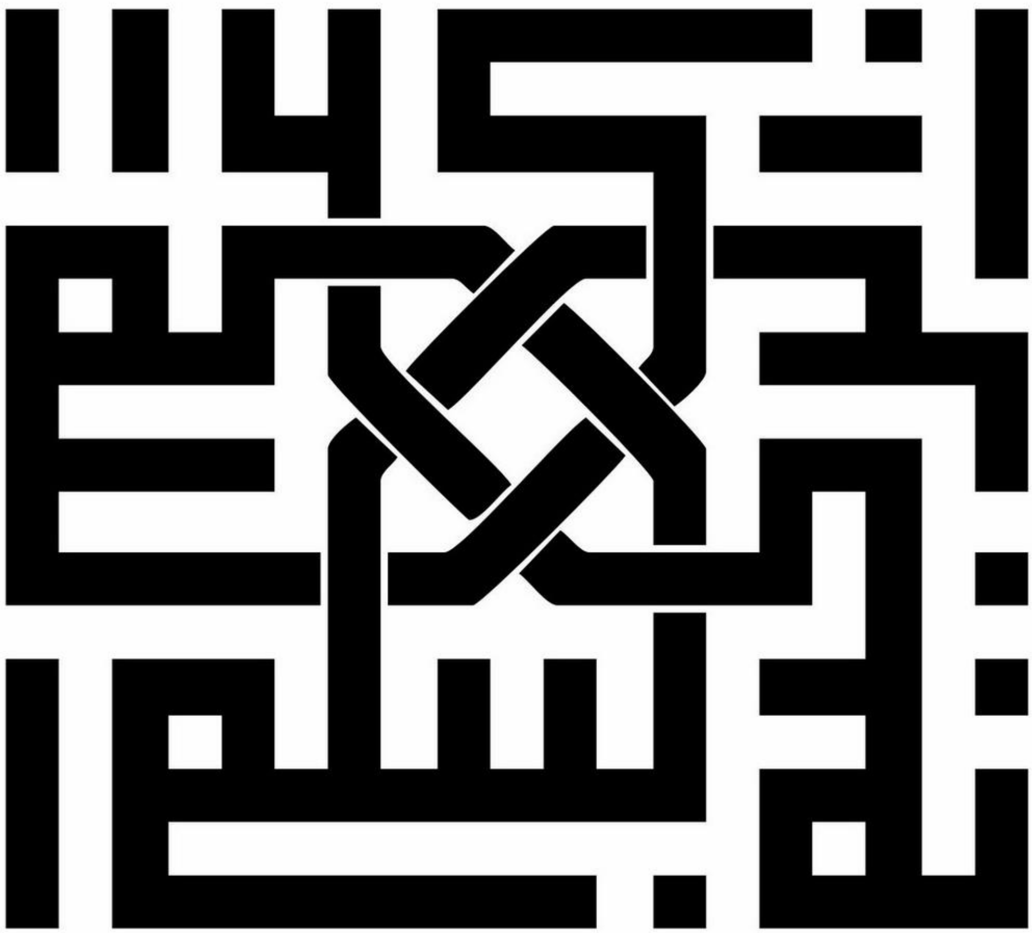


دانشگاه صنعتی اصفهان

دانشکده‌ی مهندسی برق و کامپیوتر

بررسی راه‌حل‌های دقیق و غیردقیق برای مسائله‌ی اشتعال گراف

گزارش پروژه‌ی کارشناسی

حدیث احمدیان

۹۶۲۲۶۱۳

استاد پروژه

دکتر حسین فلسفین

شهریور ۱۴۰۰

فهرست مطالب

چکیده

هدف از نگارش این گزارش معرفی مسئله‌ی اشتعال گراف، به‌عنوان یک مدل از انتشار اطلاعات در شبکه‌های اجتماعی و تلاش برای حل آن با رویکردهای دقیق و غیردقیق بوده است. این گزارش در هشت فصل تنظیم شده است که در طی آن ها مسئله معرفی شده و سعی می‌شود رویکردهای مختلف برای حل آن پیشنهاد و پیاده‌سازی شود. هم چنین در پایان گزارش، راه‌حل‌های پیاده‌سازی شده روی تعداد قابل قبولی نمونه تست شده و نتایج آن ها از لحاظ سازگار بودن و مدت‌زمان حل، با یکدیگر مقایسه شده‌اند.

فصل اول: مقدمه

امروزه، گسترش شبکه‌های اجتماعی[1] و تأثیر آن بر تغییر سبک ارتباطات بر کسی پوشیده نیست. به دنبال این پدیده، نه‌تنها نوع ارتباطات تغییر کرده، بلکه مفاهیم گسترده‌ای را پدید آورده که پیش‌ازاین وجود نداشت. یکی از این مفاهیم، سرعت گسترش اطلاعات[2] در شبکه‌های ارتباطی است. منظور از یک شبکه‌ی ارتباطی، مجموعه‌ای از افراد یا حساب‌های کاربری در شبکه‌های اجتماعی هستند که با یکدیگر ارتباطات دارند و منظور از گسترش اطلاعات، رسیدن اطلاعات به این افراد در طول زمان است. مفهوم گسترش اطلاعات در دنیای امروز توجه زیادی را به خود جلب کرده، چه از نظر بررسی‌های آماری و چه برای اهدافی مثل پخش کردن اطلاعات تبلیغاتی. ازاین‌رو این به نظر می‌رسد بررسی و درک این مفهوم برای بسیاری از کاربردها و اهداف مهم و حتی ضروری باشد. ازاین‌رو، مانند هر مسئله‌ی دیگری در دنیای واقعی، به کمک مدل‌سازی مسئله در قالب مدل‌های ریاضی، می‌توان دید خوبی برای بررسی و حل مسئله به دست آورد.

گراف[3]ها همواره ساختارهای مفیدی برای مدل‌سازی و حل مسائل بوده‌اند و در طی سال‌ها، مفاهیم زیادی به این شکل تعریف و بررسی شده‌اند. مفهوم بیان شده نیز از این قاعده مستثنی نیست. یک دیدگاه ساده به شبکه‌های ارتباطی،

[1] Social media
[2] Data spread
[3] Graph

مدل کردن آن به شکل یک گراف است. اگر فرض کنیم هر شخص در این شبکه‌ی ارتباطی یک نود است و هر ارتباط بین افراد یال است، می‌توان گسترش اطلاعات را نیز بر اساس این گراف این مدل‌سازی تعریف کرد، به این شکل که هر نود که اطلاعاتی در دست دارد، می‌تواند آن را با نود همسایه‌ی خود به اشتراک بگذارد و به این شکل همسایه‌اش نیز اطلاعات را در اختیار خواهد داشت. به این شکل می‌توانیم مفهومی که تا به اینجا یک مفهوم کلی و شهودی داشت را به مسئله‌ای تبدیل کنیم که مؤلفه‌های آن مشخص است و از راه‌های شناخته شده قابل بحث و حل است. مسئله‌ی مدل‌سازی شده در بستر این پروژه، مسئله‌ی اشتعال گراف[۴] نام دارد که یک مسئله‌ی NP-سخت[۵] است.

این گزارش در هشت فصل تنظیم شده است. در فصل دوم مسئله‌ی اشتعال گراف را به طور دقیق شرح می‌دهیم و چند مثال از مسئله را بررسی می‌کنیم. سه راه حل دقیق برای حل این مسئله پیشنهاد شده است. راه حل اول که ابتدایی‌ترین رویکرد استفاده شده برای حل مسئله است، استفاده از یک جستجوی عقب‌گرد[۶] است که در آن سعی شده با توجه به شرایط مسئله، برای کاهش فضای جستجو، از روش‌های ابتکاری[۷] استفاده شود و با توجه به کران‌های پیدا شده برای پاسخ مسئله، فضای جستجو محدود شود. در فصل سوم به این راه حل پرداخته شده و پیاده‌سازی می‌شود.

در ادامه به راه حل‌های پیشرفته‌تر می‌پردازیم. دومین راه حل، حل مسئله در قالب یک مسئله‌ی بهینه‌سازی مقید[۸] است. در این روش، سعی می‌شود مسئله با تعدادی از قید روی متغیرهایی با دامنه‌های مشخص و گسسته تعریف شود، و سعی می‌شود متغیرها به گونه‌ای مقداردهی شوند که ضمن ارضای قیود، تابع هدف[۹] مورد نظر بهینه شود. در فصل چهارم به این راه‌حل می‌پردازیم و آن را پیاده‌سازی می‌کنیم. در فصل پنجم، سعی می‌کنیم مسئله را به همان روش ولی تنها با استفاده از قیدهای خطی و تابع هدف خطی بیان کنیم. که این روش حل مسئله، برنامه‌ریزی خطی عدد صحیح نام دارد و در انتها به پیاده‌سازی آن می‌پردازیم. در فصل ششم این راه حل ها را روی تعدادی گراف بررسی خواهیم کرد و با یکدیگر مقایسه می‌کنیم.

در فصل هفتم سعی خواهیم کرد در کنار روش‌های دقیق مطرح شده، از روش غیردقیق الگوریتم ژنتیک[۱۰] نیز برای حل مسئله استفاده کنیم. سرانجام در فصل هشتم، به جمع‌بندی و نتیجه‌گیری نهایی خواهیم پرداخت.

---

[۴] Graph burning
[۵] NP-hard
[۶] Back track
[۷] heuristic
[۸] constrained optimization problem
[۹] objective function
[۱۰] Genetic algorithm

فصل دوم: معرفی مسئله‌ی اشتعال گراف

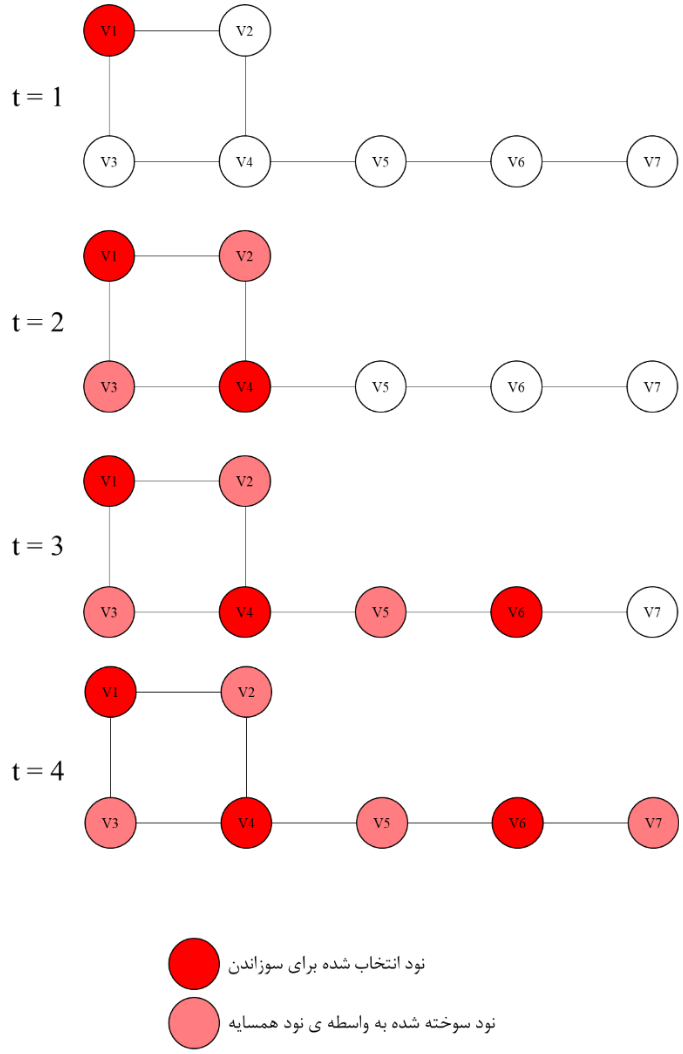
در این فصل قصد داریم ابتدا مسئله‌ی اشتعال گراف را به عنوان یک مدل برای انتشار اطلاعات در شبکه‌های اجتماعی طبق [1] معرفی کنیم، و با تعریف و جزئیات آن آشنا شویم. در ادامه، درمورد کلاس سختی و دسته‌بندی مسئله بحث خواهیم کرد، و در انتها نیز تلاش می‌کنیم با آوردن نمونه‌های خاص از این مسئله و نمایش راه حل آن‌ها به تبیین مسئله کمک کنیم.

۱-۲- معرفی مسئله

مسئله‌ی اشتعال گراف به عنوان یک مدل برای انتشار اطلاعات در شبکه‌های اجتماعی روی یک گراف بدون جهت و بدون وزن ساده تعریف می‌شود. درواقع اگر هر شخص در یک شبکه‌ی ارتباطی یک نود درنظر گرفته شود، ارتباط بین آن‌ها یک یال بین آن دو نود خواهد بود. منظور از ارتباط این است که آن افراد برای یکدیگر اطلاعات ارسال می‌کنند. حال اگر اطلاعاتی در اختیار یک نود باشد، آن نود می‌تواند با نودهایی که با آن ها مرتبط است اطلاعات به اشتراک بگذارد. اگر

این انتشار اطلاعات را مثل انتشار آتش در گراف در نظر بگیریم، هر نودی که اطلاعات به آن رسیده باشد خواهد سوخت و می‌تواند همسایه‌های خود را بسوزاند، به این ترتیب مسئله‌ی اشتعال گراف تعریف می‌شود:

V را مجموعه‌ی رأس‌ها و E را مجموعه‌ی یال‌های گراف G در نظر می‌گیریم. در هر مرحله‌ی زمانی $t > 0$ می‌توانیم یک نود را برای سوختن انتخاب کنیم. به نودی که برای سوختن انتخاب شده است منبع[۱۱] گفته می‌شود. در مرحله‌ی زمانی $t + 1$ تمام همسایگان نود منبع و هم چنین همسایگان تمام نودهای از قبل سوخته شده، می‌سوزند و یک منبع جدید از بین نودها نیز انتخاب خواهد شد، و این مراحل به همین شکل ادامه خواهند یافت. به این ترتیب، در لحظه‌ای از زمان تمامی نودها خواهند سوخت. ترتیب منبع‌هایی که برای سوزاندن در هر مرحله انتخاب کرده‌ایم، بر تعداد مراحل لازم برای سوختن کل نودهای گراف تأثیرگذار است؛ بنابراین وابسته به ترتیب دنباله‌ای از نودها که برای سوزاندن انتخاب می‌کنیم، مراحل لازم برای سوختن تمام نودها متغیر خواهد بود. پس حالتی خاص از این دنباله وجود دارد که تعداد این

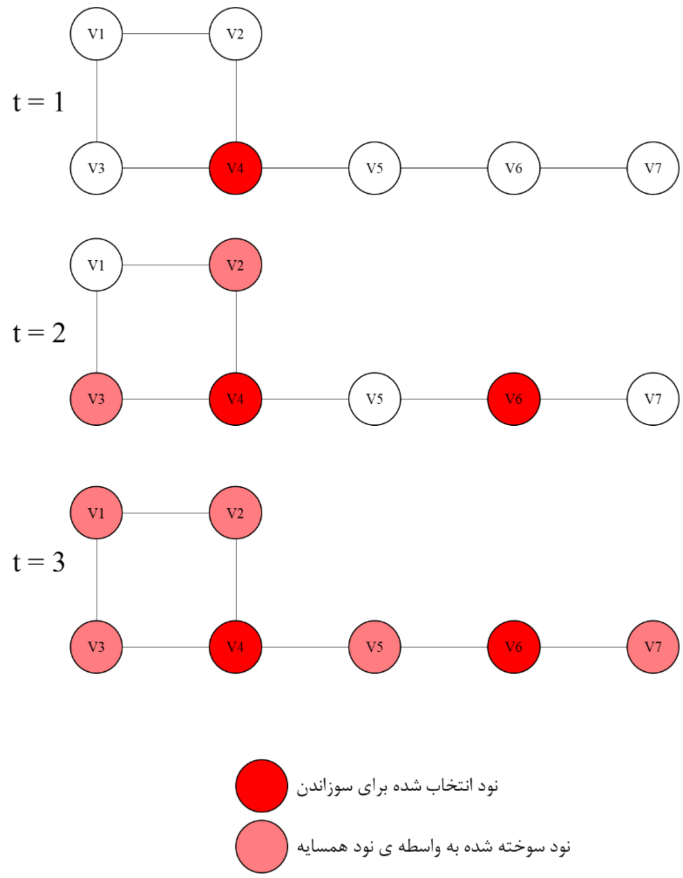


شکل ۲-۲- سوختن گراف با دنباله ی غیر بهنیه‌ی v4,v1,v6

۱۱ source

مراحل را کمینه می‌کند. به این تعداد کمینه، عدد اشتعال[۱۲] گفته می‌شود و با نماد $b(G)$ نشان داده می‌شود. منظور از حل مسئله‌ی اشتعال گراف، درواقع یافتن این میزان کمینه است.

شایان ذکر است که دنباله‌ای که تعداد مراحل سوختن را کمینه می‌کند لزوماً یکتا نیست و ممکن است تعداد زیادی دنباله‌ی متفاوت وجود داشته باشد که منجر به میزان کمینه شود. در ادامه روی گراف نمونه‌ی زیر دو مسیر متفاوت برای سوختن گراف را خواهیم دید که یکی از آن‌ها منجر به میزان کمینه‌ی $b(G) = 3$ می‌شود.



شکل ۱-۲- سوختن گراف با دنباله ی بهنیه‌ی v4,v6

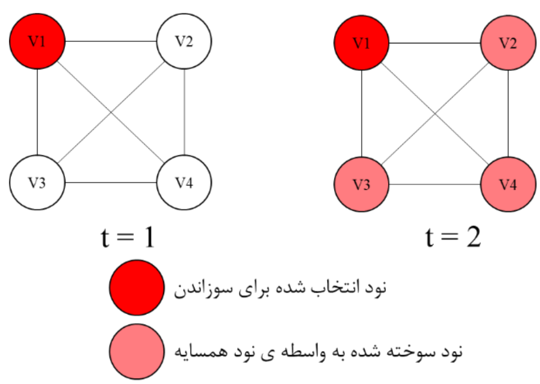
[۱۲] Burning number

۲-۲- پیچیدگی مسئله و نمونه های خاص

علی رغم اینکه مسئله‌ی اشتعال گراف به طور کلی NP-سخت است، نمونه‌های خاصی از آن وجود دارند که در زمان چند جمله‌ای قابل حل هستند، یا برای خانواده‌های خاصی از گراف مقادیر صریح عدد اشتعال بدون نیاز به جستجو قابل به دست آمدن است. در این قسمت در مورد برخی از این نمونه‌های خاص صحبت می‌کنیم

۲-۲-۱- گراف کامل[۱۳]

گراف کامل از گراف‌هایی هستند که عدد اشتعال آن‌ها بدون نیاز به حل مشخص است. می‌دانیم در گراف کامل بین هر دو رأس یال وجود دارد بنابراین اگر در $t = 1$ یکی از این نودها را به عنوان منبع بسوزانیم، در $t = 2$ تمام نودها خواهند سوخت زیرا بین نود منبع و تمام نودهای دیگر یال وجود دارد. پس صرف نظر از تعداد یال‌ها می‌توان نتیجه گرفت عدد اشتعال گراف‌های کامل ۲ است.



شکل .Type equation here۲-۳- مراحل سوختن یک گراف کامل با ۴ نود

۲-۲-۲- سایر گراف‌ها

علاوه بر گراف‌های کامل، نمونه‌های دیگری از گراف وجود دارند که یافتن راه حل آن ها نسبت به حالت کلی سریع‌تر است، با توجه به [2]، چند نمونه‌ی دیگر از گراف‌ها هستند که مقدار دقیق عدد اشتعال آن‌ها بدون نیاز به حل، قابل محاسبه است این گراف‌ها عبارت‌اند از گراف پترسن[۱۴]، گراف تتا[۱۵]، گراف‌های شبکه‌ای[۱۶] و ... .

---

[۱۳] Complete graph
[۱۴] Petersen graph
[۱۵] Theta graph
[۱۶] Grid graph

همچنین گراف‌هایی وجود دارند که یافتن عدد اشتعال آنها به کلاس پیچیدگی NP-سخت تعلق ندارد. دسته‌ی اول از این گراف‌ها، گراف‌هایی هستند که برای یافتن عدد اشتعال آنها الگوریتم‌هایی با راه حل چندجمله‌ای وجود دارد. این گراف‌ها عبارت‌اند از گراف‌های عنکبوتی[۱۷] و مسیر-جنگل[۱۸] ها. دسته‌ی بعدی نیز گراف‌هایی هستند که حل مسئله‌ی اشتعال گراف در آن ها NP-کامل است و نه NP-سخت. درخت‌هایی[۱۹] با ماکزیمم درجه ۳ از این دسته هستند.

---

[۱۷] Spider graph

[۱۸] Path-forest

[۱۹] tree

فصل سوم: حل مسئله با جستجوی عقب‌گرد

حال که درمورد مسئله‌ی اشتعال گراف اطلاعات کافی به دست آوردیم، باید به رویکردهای ممکن برای حل آن فکر کنیم. اولین رویکردی که به طور طبیعی برای حل مسائل، پیش روی ماست، جستجوی کامل فضای حالت است. در این رویکرد سعی می‌شود با بررسی تمام حالات ممکن به جواب بهینه‌ی مورد نظر برسیم. در این فصل سعی می‌کنیم ضمن پیاده‌سازی این راه حل، با استفاده از کران‌های ممکن و استفاده از هیوریستیک‌های مناسب، این راه حل را بهبود بخشیم.

۱-۳- جستجوی عقب‌گرد

شیوه‌ی جستجوی عقب‌گرد یک روش حل مسئله با استفاده از جستجو در فضای حالت است که پیش از گسترش یک گره در فضای حالت، با روش‌هایی بررسی می‌کند که آیا آن گره امیدبخش[۲۰] هست و منجر به جواب می‌شود یا نه. در واقع این روش درخت فضای مسئله را ایجاد کرده و تعیین می‌کند کدام گره‌ها برای رسیدن به پاسخ امیدبخش هستند، و بدین

[۲۰] Promising

ترتیب با گسترش ندادن سایر گره‌ها، می‌تواند به طور میانگین در زمان کمتری به پاسخ برسد. عقب‌گرد در واقع حالت اصلاح شده‌ی جستجوی عمق اول[۲۱] است.

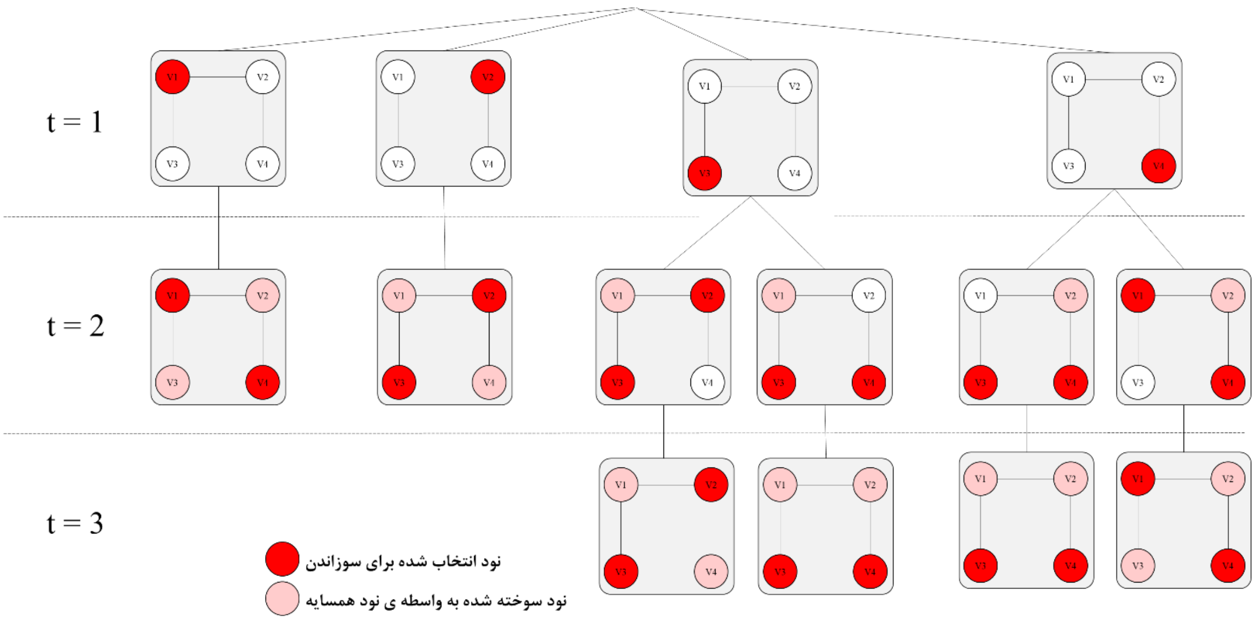
یک روش اصولی برای بررسی امیدبخش بودن یا نبودن یک گره و انتخاب آن برای گسترش، این است که بررسی کنیم آیا بهترین جوابی که ممکن است این گره به آن برسد، از بهترین جوابی که تاکنون یافت شده بهتر هست یا نه. بدیهی است اگر بهترین جوابی که یک گره، ممکن است در آینده به آن برسد، از بهترین جوابی که تا کنون پیدا شده بدتر باشد، ارزش گسترش ندارد و منجر به یافتن جواب بهینه‌ی مسئله نخواهد شد.

حال می‌خواهیم رویکرد توضیح داده شده را برای مسئله‌ی اشتعال گراف در پیش بگیریم. در ابتدا باید فضای حالت و طریقه‌ی جستجوی عمق اول برای این مسئله را تبیین کنیم. فضای حالت مسئله در واقع ترتیب‌های مختلفی از نودها خواهند بود که قرار است هر بازه‌ی زمانی برای سوزاندن انتخاب شوند. وقتی همسایه‌های نودهای سوزانده شده را می‌سوزانیم و نود انتخاب شده‌ی بعدی را نیز می‌سوزانیم، در نهایت، در مرحله‌ای از زمان تمام نودها خواهند سوخت. در این جستجو ما به دنبال ترتیبی از انتخاب هستیم که تعداد مراحل برای سوختن کل گراف را کمینه کند. به این ترتیب برای جستجو در هر مرحله، تمام همسایه‌های نودهای سوخته شده را می‌سوزانیم و یک نود از میان نودهای تابه‌حال سوخته نشده انتخاب می‌کنیم و به مرحله‌ی بعد می‌رویم و به این شکل درخت جستجو را تشکیل می‌دهیم. این رویکرد را به صورت عمق اول ادامه می‌دهیم و در عمقی از جستجو که تمام نودها در آن سوخته شوند، به یک راه حل خواهیم رسید که تعداد مراحل طی شده تا آن، عدد اشتعالی است که تابه‌حال به دست آورده‌ایم.

---

[۲۱] Depth first search

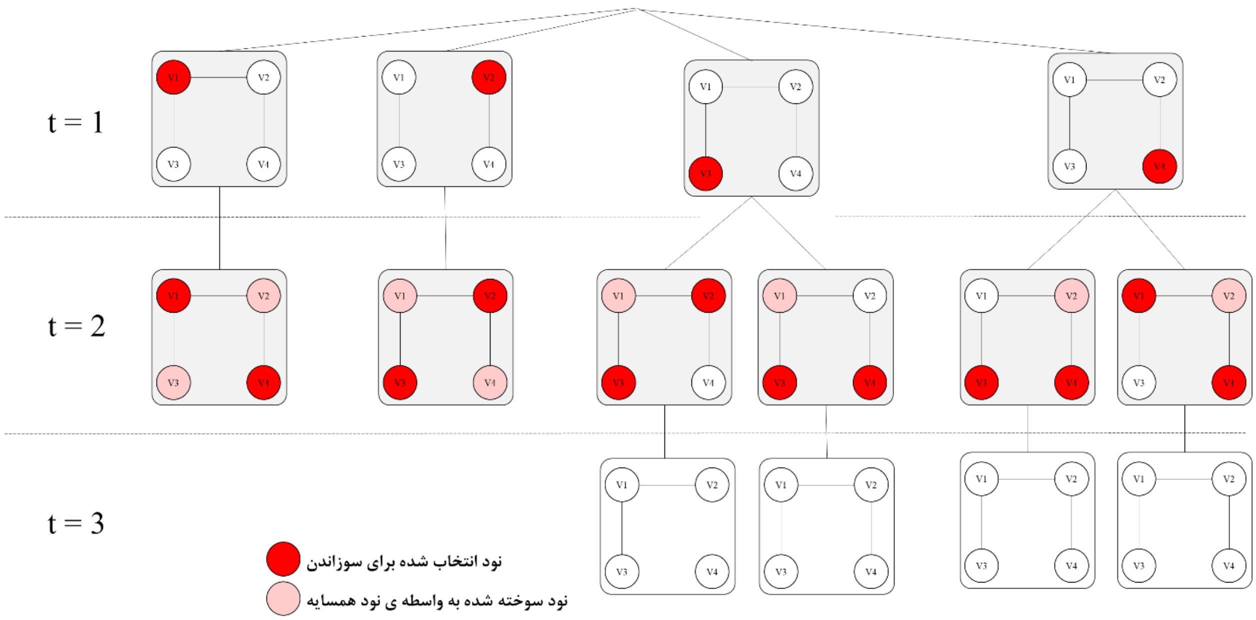
واضح است که امکان آن وجود دارد که این عدد تا اینجا بهینه نباشد. به همین ترتیب جستجوی عمق اول را ادامه می‌دهیم و هر گاه راه حل بهتری از بهترین راه حل تا آن مرحله پیدا کردیم، به عنوان بهترین راه حل تا آن مرحله ذخیره می‌کنیم. در شکل زیر چگونگی جستجو برای یک گراف ساده را می‌بینیم.



شکل ۱-۳- مراحل جستجوی عمق اول برای مسئله

برای پیاده‌سازی رویکرد عقب‌گرد در حل مسئله‌ی اشتعال گراف باید گره‌هایی که در طی جستجو امیدبخش نیستند را گسترش ندهیم، به همین دلیل اگر کمترین تعداد مراحل اشتعال پیدا شده، برای سوزاندن یک گراف تا کنون x باشد، وقتی به گره‌ای برخورد کنیم که y مرحله برای رسیدن به آن صرف شده ولی تمام نودها نسوخته باشند، اگر $y \geq x$ آنگاه این گره ارزش گسترش نخواهد داشت، زیرا قطعاً به پاسخی بهتر از بهترین پاسخ تا کنون یافت شده، نخواهد رسید.

در همان مثال مطرح شده در بالا، می‌توان دید با اجرای تکنیک توضیح داده شده، فضای کمتری جستجو خواهد شد و گره‌هایی که نتیجه بخش نیستند گسترش نخواهند یافت. بخش هایی که بی رنگ نشان داده شده اند، جستجو نخواهند شد.



شکل ۲-۳- مراحل جستجوی عقب‌گرد برای مسئله

در ادامه روش‌هایی را برای بهبود این راه حل استفاده شده را مورد بحث قرار می‌دهیم.

۱-۱-۳- کران‌ها

علاوه بر تکنیک توضیح داده شده در قسمت قبل، عدد اشتعال گراف‌ها، کران‌هایی دارد که با توجه به آن‌ها می‌توان فهمید گسترش یک گره حین جستجو امیدبخش است یا نه و بدین ترتیب شاخه‌هایی که به جواب نمی‌رسند را هرس[۲۲] کرد. در ابتدا توضیح می‌دهیم چگونه با دانستن یک کران می‌توان راه حل مطرح شده در قسمت قبل را بهبود بخشید و بعد از آن کران‌های موجود برای عدد اشتعال گراف و استفاده‌ای که از آن در پیاده‌سازی راه حل شده است را توضیح خواهیم داد.

فرض کنید میدانیم x یک کران بالا برای عدد اشتعال گراف G است، این بدان معناست که عدد اشتعال G هرگز از x بیشتر نخواهد شد و همواره $b(G) \leq x$ خواهد بود. به این ترتیب در حین جستجو، اگر در شاخه‌ی جاری جستجو به گره‌ای برسیم که y مرحله را طی کرده است ولی هنوز تمام نودهای G نسوخته باشد و $y > x$ آن گاه، گره ارزش

[۲۲] prune

گسترش را نخواهد داشت. دلیل این امر نیز بدیهی است، چون $y > x$ و $b(G) \leq x$ بنابراین $b(G) < y$ ، پس شاخه‌ی جاری جستجو در مراحلی بیش از مراحل بهینه گراف را می‌سوزاند که این مطلوب ما نیست.

طبق [3] یک کران اثبات شده برای عدد اشتعال گراف، $b(G) \leq r + 1$ است، که بیان می‌دارد عدد اشتعال گراف همواره از قطر[23] گراف به علاوه‌ی یک، کمتر یا مساوی خواهد بود. منظور از قطر گراف برابر است با اندازه‌ی بیشترین خروج از مرکز[24]. اگر فاصله‌ی بین دو رأس در گراف را طول کوتاه‌ترین مسیر بین آن دو در نظر بگیریم، خروج از مرکز هر راس به معنای حداکثر فاصله آن با رأس با سایر رئوس است. به بیان دیگر، منظور از قطر گراف، طول بلندترین مسیر در بین تمام کوتاه‌ترین مسیرهاست.

همان‌طور که توضیح داده شد، می‌توان از این کران بالا برای هرس کردن شاخه‌های جستجو استفاده کرد. پس قبل از شروع جستجو کافی است قطر گراف را محاسبه کنیم و شاخه‌هایی که در عمق بیشتر از $r + 1$ به جواب نمی‌رسند را هرس کنیم که از این روش در پیاده‌سازی استفاده شده است. نکته‌ی قابل‌توجه این است که اگر بخواهیم مسئله‌ی سوختن گراف را روی گراف‌های ناهمبند[25] نیز تعریف کنیم، در استفاده از این کران به مشکل برمی‌خوریم زیرا در گراف ناهمبند بین همه‌ی رأس ها مسیر وجود ندارد. می‌توانیم طول مسیرهایی که وجود ندارند را بینهایت فرض کنیم و به این ترتیب قطر گراف نیز بینهایت خواهد شد و شاخه‌ای هرس نخواهد شد. یک جایگزین دیگر نیز این است که کران بالا را به اندازه‌ی تعداد نودها در نظر بگیریم زیرا اگر گراف هیچ یالی نیز نداشته باشد و تعداد نودها n باشد با هر بار سوزاندن یکی از این نودها نهایتاً گراف در n مرحله خواهد سوخت.

از این کران‌ها برای هرس کردن شاخه‌های جستجو در پیاده‌سازی استفاده شده و مورد ناهمبند بودن گراف نیز در پیاده‌سازی پشتیبانی شده است، که در بخش‌های بعدی درمورد آن صحبت خواهد شد.

۲-۱-۳- هیوریستیک درجه‌ی نود[26] و سایر بهبودها

در قسمت قبل دیدیم که با هرس کردن شاخه‌هایی که بهترین پاسخ مورد انتظار آن‌ها از بهترین پاسخ تاکنون یافت شده، بهتر نیست، می‌توانیم زمان جستجو را کاهش دهیم. این که این رویکرد چقدر در کاهش فضای جستجو مؤثر است، وابسته به این است که چقدر از شاخه‌ها هرس می‌شوند. هرس شدن شاخه‌ها نیز وابسته به ترتیبی است که شاخه‌ها جستجو می‌شوند. به عنوان توضیح بیشتر، اگر شاخه‌ای که زودتر دیده می‌شود، جواب بهتری نسبت به تعداد قابل توجهی از

[23] Diameter
[24] Eccentricity
[25] Unconnected
[26] Node degree

شاخه‌های بعدی داشته باشد، تعداد زیادی شاخه‌ی غیرامیدبخش در ادامه هرس خواهد شد، اما اگر ابتدا شاخه‌هایی با پاسخ‌های بدتر دیده شوند، طبیعتاً شاخه‌های بعدی که جواب بهتری دارند هرس نخواهند شد، پس یافتن سریع‌تر یک جواب خوب حائز اهمیت است.

به منظور ایجاد یک ترتیب مناسب که به هرس شدن بهتر شاخه‌ها کمک می‌کند، می‌توان از یک هیوریستیک مناسب استفاده کرد که کمک کند شاخه‌ها با جواب بهتر، زودتر از شاخه‌ها با جواب بدتر قرار بگیرند. روشی که برای مسئله‌ی اشتعال گراف مناسب به نظر می‌رسد، دادن اولویت سوختن به رئوسی است که درجه‌ی بیشتری دارند. درجه‌ی یک رأس برابر است با تعداد همسایه‌های آن و چون در مرحله از مسئله‌ی اشتعال گراف، همسایه‌ی تمام نودهای از قبل سوخته شده می‌سوزند، به طور شهودی به نظر می‌رسد که احتمالاً اولویت دادن به نودها با درجه‌ی بالاتر منجر به سریع‌تر سوختن گراف شود. باید توجه شود این یک دیدگاه شهودی و تقریبی است و اگر همواره انتخاب نود با درجه‌ی بالاتر به راه حل بهتری منجر می‌شد، مسئله‌یک راه حل حریصانه‌ی[۲۷] چندجمله‌ای می‌داشت که در آن کافی بود نودها فقط به ترتیب درجه برای سوختن انتخاب شود، که می‌دانیم این‌طور نیست. طی استفاده از این هیوریستیک در پیاده‌سازی، تأثیر مثبت آن در کاهش زمان جستجو، به خصوص در انتخاب نودها با درجه‌های بالاتر در مراحل ابتدایی سوختن، قابل مشاهده بود که در قسمت‌های بعدی درمورد پیاده‌سازی نهایی بیشتر بحث خواهد شد. شایان ذکر است علاوه بر استفاده از کران‌ها و اولویت دادن به درجه‌ی نود، در پیاده‌سازی به حالت خاص گراف کامل نیز توجه شده و در صورتی که تشخیص داده شود گراف کامل است، بدون نیاز به جستجو عدد ۲ به عنوان عدد اشتعال گراف اعلام می‌شود.

[۲۷] Greedy

۲-۳- پیاده‌سازی

حال که درمورد رویکرد کلی راه حل اولیه و بهبودهای آن صحبت کردیم، می‌توانیم آن را پیاده‌سازی کنیم. باید اشاره شود این پیاده‌سازی با استفاده از زبان پایتون و ابزار ژوپیتر نوت بوک[۲۸] انجام شده است. برای استفاده از توابع مورد نیاز گراف‌ها، از کتاب‌خانه‌ی networkx پایتون کمک گرفته شده و فرمت ورودی گراف‌ها برای این پیاده‌سازی، طبق استاندارد DIMACS[۲۹] در نظر گرفته شده.

در قطعه کد زیر تابع اصلی پیاده‌سازی آورده شده است، که توضیح داده خواهد شد. سایر بخش های کد از جمله توابع خواندن گراف‌ها از ورودی و نوشتن نتایج درخروجی، کار با گراف مانند محاسبه‌ی قطر یا مرتب کردن نودهای گراف بر حسب درجه و ... برای جلوگیری از طولانی شدن گزارش آورده نشده.

```python
def dfs_search(seq,burned,t,m,best,G1,r,sorted_nodes):

    if t>best[0][0] | t>r+1:
        return

    for i in range(len(burned)):
        neighbors = list(G1.neighbors(str(burned[i])))
        neighbors = [int(x) for x in neighbors]
        burned.extend(neighbors)
        burned=list(dict.fromkeys(burned))

    if len(burned)==m or len(burned)==m-1:
        if(t<best[0][0]):
            best[0][0]=t
            best[1]=seq[:]
        return


    else:
        for i in list(sorted_nodes):
            if i not in burned:
                next_burned=burned[:]
                next_burned.append(int(i))
                next_seq=seq[:]
                next_seq.append(int(i))
                dfs_search(next_seq,next_burned,t+1,m,best,G1,r,sorted_nodes)
```

قطعه کد ۱-۳- تابع اصلی پیاده‌سازی جستجوی عقب‌گرد

---

[۲۸] Jupyter notebook

[۲۹] Discrete Mathematics and Theoretical Computer Science center

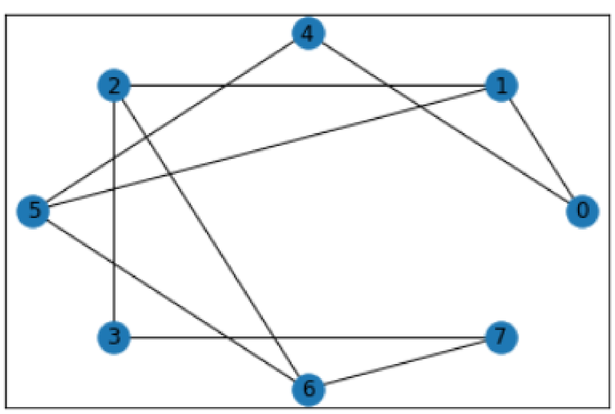
الف) ورودی‌ها:

- Seq: دنباله‌ی انتخاب نودها در شاخه‌ی جاری جستجو با مقدار اولیه‌ی تهی.
- Burned: نودهای سوخته شده تا این مرحله با مقدار اولیه‌ی تهی.
- t: تعداد مراحل طی شده.
- m: تعداد نودهای گراف.
- best: بهترین راه حل پیدا شده تا به‌حال.
- G1: گراف ورودی (در اینجا گراف از فرمت DIMACS خوانده شده و به فرمت networkx تغییر داده شده).
- r: قطر گراف (محاسبه شده از قبل).
- Sorted_nodes: نودهای مرتب شده بر اساس درجه که قبل از فراخوانی تابع محاسبه شده، تا طبق هیوریستیک توضیح داده شده، به ترتیب درجه برای جستجو انتخاب شوند.

ب) خروجی: هنگامی که تابع جستجو را به پایان رساند و متوقف شد، مقدار بهینه‌ی عدد اشتعال و دنباله‌ی بهینه ی انتخاب نودها در متغیر best خواهد بود.

ج) روند کار: در ابتدا با مقایسه‌ی مراحل طی شده با کران‌هایی که قبلاً بحث شد، بررسی می‌شود که آیا ادامه‌ی این شاخه ی جستجو امیدبخش هست یا نه. اگر امیدبخش بود، تمام همسایگان نودهای تابحال سوخته شده به لیست نودهای سوخته شده افزوده می‌شوند و سپس اگر سوختن گراف به پایان رسیده بود و راه حل این شاخه از بهترین راه حل تا کنون بهتر بود، راه حل جدید به عنوان بهترین ذخیره می‌شود. اگر سوختن به پایان نرسیده بود، تابع نود بعدی را برای سوختن انتخاب کرده و به طور بازگشتی وارد مرحله ی بعدی خواهد شد.

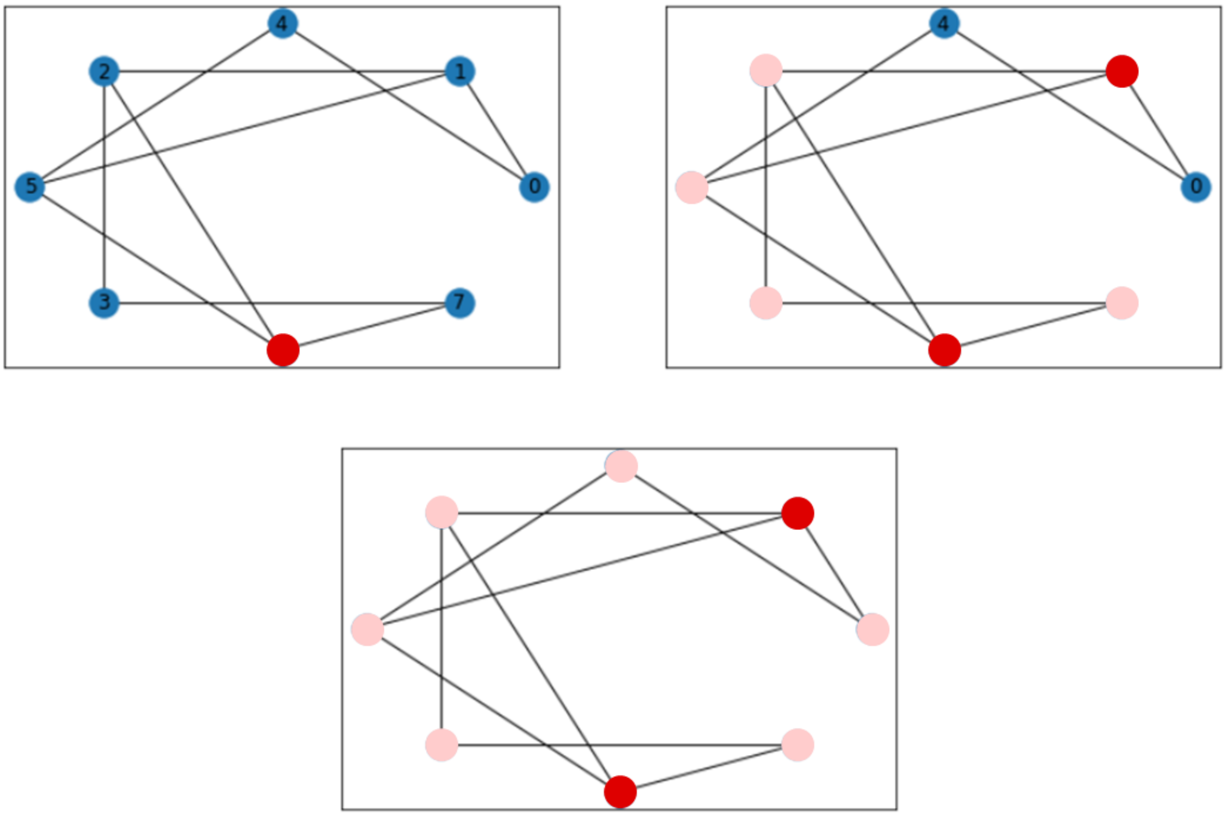
در زیر یک نمونه ی ورودی و خروجی از این پیاده‌سازی را خواهیم دید.



شکل ۳-۳- ورودی پیاده‌سازی عقب‌گرد

```
b(G):3
time(s):0.0009970664978027344
burning sequence :
[6, 1]
```

شکل ۴-۳- خروجی پیاده سازی عقب‌گرد



شکل ۵-۳- نمایش راه حل حاصل از جستجوی عقب‌گرد

فصل چهارم: حل مسئله از طریق مدلسازی به یک مسئله بهینه‌سازی مقید

در فصل قبل، ما مسئله را به وسیله‌ی یک جستجوی عقب‌گرد بهبودیافته حل کردیم. اما این تنها روش ممکن برای حل مسئله‌ی اشتعال گراف نیست. یک رویکرد مناسب برای حل مسائل بهینه‌سازی، مدل‌سازی آن‌ها در بستر یک مسئله‌ی بهینه‌سازی مقید است، که در آن تلاش می‌کنیم مسئله را در قالب چند قید بیان کنیم که مقداردهی ارضاکننده‌ی قیود به طوری که تابع هدف بهینه شود، راه حل را به ما می‌دهد. در این فصل قصد داریم مسئله‌ی اشتعال گراف را در قالب یک مسئله‌ی بهینه‌سازی مقید (cop) بیان کنیم، و سپس با ابزارهای موجود، آن را پیاده‌سازی و حل کنیم.

۴-۱- بیان مسئله اشتعال گراف در بستر مسئله‌ی بهینه‌سازی مقید (cop)

لازم است در ابتدا به طور اجمالی درمورد مسائل بهینه‌سازی مقید صحبت شود. مسائل بهینه‌سازی مقید زیرمجموعه‌ای از مسئله‌های بهینه‌سازی ریاضیاتی هستند که در آن‌ها دسته‌ای از متغیرها با دامنه‌ی مشخص وجود دارد، چند قید روی این متغیرها تعریف شده است که هر مقداردهی به متغیرها باید تمامی قیدها را ارضا کند، البته در بعضی موارد خاص لزومی به ارضای حتمی بعضی از قیدها نیست، که این موضوع بحث ما نیست. تفاوت یک مسئله‌ی بهینه‌سازی مقید با یک

مسئله‌ی ارضای محدودیت[30] ساده در وجود یک تابع هدف است. در یک مسئله‌ی بهینه‌سازی مقید، علاوه بر ارضای قیدها، مقداردهی ما به متغیرها باید یک تابع هدف تعریف شده روی متغیرها را، بهینه کند. هدف از این بهینه‌سازی می‌تواند متفاوت باشد. برای مثال اگر تابع هدف نمایانگر درآمد و امتیاز کسب شده در یک مسئله باشد، بیشینه شدن آن مطلوب است و اگر نماینده‌ی انرژی هدررفته باشد، کمینه شدنش مورد نظر است.

مسائل زیادی هستند که قابلیت بیان شدن به شکل یک مسئله‌ی بهینه‌سازی را دارند. بیان مسائل به این شکل مزیت‌هایی دارد. از جمله این که ابزارها و روش‌های حل بسیار کارآمدی در این زمینه توسعه داده شده که می‌تواند در سرعت حل مسئله بسیار مؤثر باشد. مسئله‌ی اشتعال گراف نیز از این قاعده مستثنی نیست و تلاش برای بیان آن به شکل یک مسئله‌ی بهینه‌سازی مقید و حل آن به وسیله‌ی ابزارها و متدهای کارآمد، می‌تواند بسیار مفید باشد. در ادامه سعی می‌کنیم با استفاده از قیدها و متغیرهایی، در قالب گفته شده، مسئله‌ی اشتعال گراف را بیان کنیم.

۱-۱-۴- متغیرها و قیدها

حال که در مورد مسائل بهینه‌سازی مقید صحبت کردیم ، باید تلاش کنیم مسئله‌ی اشتعال گراف را نیز به این شکل بیان کنیم. برای این کار ابتدا باید موجودیت‌های اصلی مسئله را بشناسیم و بر حسب آن، متغیرها را تعریف کرده، و با توجه به هدفی که در مسئله می‌خواهیم به آن برسیم، قیدها را تعریف کنیم. قیدها و متغیرهای نوشته شده برای توصیف مسئله به شکل زیر است . توجه کنید که n و D پارامتر[31] هستند و در ادامه بیشتر درمورد آن‌ها توضیح داده می‌شود :

$$n$$
$$D[i][j] \quad 1 \le i,j \le n$$

$$b \qquad\qquad 2 \le b \le n$$
$$T[i] \quad 1 \le i \le n \qquad 1 \le T[i] \le n$$

$$minimize\ b$$
$$subject\ to:$$
$$\forall\, j\ \ 1 \le i \le n\ (\exists\, i\ |1 \le j \le n\ and\ T[i] \le b\ and\ D[i][j] \le b - T[i])$$
$$Alldiff(T)$$

[30] Constraint satisfaction problem
[31] Parameter

در ادامه لازم است متغیرها و قیدهای روی آن‌ها توضیح داده شود:

الف) متغیرها

- در طرح مسئله دیدیم که در هر مرحله علاوه بر اینکه تمام همسایه‌های نودهای از پیش سوخته شده می‌سوزند، یک نود جدید نیز برای سوختن انتخاب می‌شود و درواقع هر راه حل مسئله متناظر است با دنباله‌ی نودهای انتخاب شده در طی مراحل. T [i] یک آرایه به طول n است که در واقع نوبت انتخاب شدن نود i ام را بیان می‌کند. مقادیری که T [i] اتخاذ می‌کند بین ۱ تا n خواهد بود، زیرا مقدار آن بیان‌گر یک نوبت برای سوختن است. برای مثال T[2]=3 بیان می‌دارد که نود شماره‌ی ۲ در سومین نوبت برای سوختن انتخاب خواهد شد.
- b معرف عدد اشتعال گراف است که قصد داریم آن را کمینه کنیم.

ب) پارامترها

- n تعداد نودهای گراف است
- D [i] [j] فاصله‌ی بین نود i ام و j است. منظور از فاصله، طول کوتاه‌ترین مسیر بین دو نود در گراف است. برای مثال اگر D[2][3]=4 به این معناست که فاصله‌ی نود شماره‌ی ۲ و نود شماره‌ی ۳ در گراف برابر ۴ است. به عبارتی D [i] [j] حاصل یافتن کوتاه‌ترین مسیر بین تمام نودها است.

پس T [i] و b قرار است پس از حل مدل مشخص و مقداردهی شوند، در حالی که n و D [i] [j] پارامتر هستند و بر حسب گراف، باید محاسبه شوند و مقدار ثابت به آن‌ها داده می‌شود.

ت) قیدها

- $Alldiff(T)$ بیان میکند که مقادیر موجود در T[i] متفاوت باشند، یعنی: $T[i] \neq T[j]\ iff\ i \neq j$ علت وجود این قید این است که T[i] نوبت انتخاب هر نود برای سوختن است و در هر نوبت فقط یک نود می‌تواند انتخاب شود پس ممکن نیست نوبت انتخاب دو نود برابر باشد.
- قید $\forall\, j\ (\exists\, i\ |1 \leq i,j \leq n\ and\ T[i] \leq b\ and\ D[i][j] \leq b - T[i])$ بیان می‌کند در انتهای b مرحله باید تمام نودها سوخته باشند. حال بیایید توضیح دهیم که این قید چگونه این موضوع را تضمین می‌کند. برای این که راحت‌تر توضیحات را دنبال کنید، فراموش نکنید که وقتی b تعداد مراحل سوختن کل گراف است یعنی ما b مرحله فرصت داریم تا کل گراف را بسوزانیم.

- $T[i] \leq b$ به این منظور است که نوبت سوخته شدن نود i باید کمتر از عدد اشتعال باشد. به بیان دیگر، وقتی عدد اشتعال b است یعنی در b مرحله باید کل گراف بسوزد، پس $T[i] \leq b$ بیان میکند که نود i در یکی از این b مرحله برای سوختن انتخاب شده است، یعنی یکی از نودهای انتخابی برای سوخته شدن بوده است.

- $D[i][j] \leq b - T[i]$ بیان میکند که فاصله ی نود i و j باید کمتر یا مساوی $b - T[i]$ باشد. این قسمت تضمین می‌کند که در صورت انتخاب شدن نود i برای سوختن در یکی از مراحل، تا قبل از پایان مراحل(مرحله ی b ام)، نود j هم خواهد سوخت. درواقع $b - T[i]$ متناظر است با تعداد مراحل باقی مانده، از زمان انتخاب i تا زمانی که به ماکزیمم مراحل ممکن (مرحله ی b ام) برسیم، یعنی به ما می‌گوید چند مرحله‌ی دیگر بعد از انتخاب i، تا انتهای مراحل باقی مانده است. چون آتش در هر مرحله، از یک نود سوخته شده، یک قدم به سمت نودهای همسایه حرکت می‌کند، وقتی فاصله‌ی نود j با نود i کمترمساوی تعداد مراحل باقی‌مانده باشد، آتش از i به j خواهد رسید.

- $\forall\, j\ (\exists\, i \mid \ldots\ldots)$ بیان می‌کند به ازای هر نود j باید یک نود i وجود داشته باشد که شرایط قید شده را ارضا کند، با توجه به شرایط توضیح داده شده، کل قید بیان می‌کند که به ازای هر نود j باید نود i ای وجود داشته باشد، که آتش در یکی از مراحل از i به j برسد و j را بسوزاند. حال اگر i=j پس نود j خودش برای سوختن انتخاب شده است و اگر i≠j نود j به واسطه‌ی همسایگی با یکی از نودهای سوخته، سوزانده شده است.

- $1 \leq i,j \leq n$ نیز مشخص است که صرفاً تعیین دامنه‌ی ممکن برای i,j است.

ث) تابع هدف: minimize b بیان می‌کند که هدف مسئله کمینه کردن تعداد مراحل اشتعال است.

۲-۴- پیاده‌سازی

همان‌طور که بیان شد ابزارهای مناسبی برای حل مسائل بهینه‌سازی مقید توسعه یافته‌اند، یکی از بهترین نمونه‌ی این ابزارها مینی‌زینک[۳۲] است که با سینتکس مخصوص، می‌توان متغیرها و قیدهای مسئله را در آن تعریف کرد و سپس با استفاده از حل‌کننده‌های مختلف راه حل بهینه را پیدا کرد. برای پیاده‌سازی مدل گفته شده در طی این پروژه دو مرحله

[۳۲] minizinc

طی شد، در ابتدا کدی به زبان پایتون برای پیش‌پردازش‌های لازم که در ادامه توضیح داده خواهد شد پیاده شد و سپس مدل نهایی در مینی‌زینک تست شد.

۱-۲-۴- پیش‌پردازش لازم

همان‌طور که در بخش توضیحات متغیرها دیدیم، لازم است برای پیاده‌سازی این راه حل، فاصله‌ی بین هر دو نود موجود در گراف را داشته باشیم. برای داشتن این فاصله‌ها لازم است قبل از پیاده‌سازی مدل، یک پیش‌پردازش روی گراف صورت گیرد و این فاصله‌ها محاسبه شود. برای این کار از کتابخانه ی networkx در پایتون و در ادیتور ژوپیتر نوت‌بوک استفاده شده است. ابتدا گراف از ورودی به فرمت DIMACS خوانده شده و سپس با تبدیل فرمت و اجرای توابع کتاب‌خانه، فاصله‌ها محاسبه شده.

علاوه بر محاسبه‌ی فاصله‌ی بین هر دو نود، به منظور سهولت در ایجاد مدل برای تعداد زیادی گراف، با استفاده از زبان پایتون، کدی به منظور خواندن مدل ها و تولید مدل نظیر آن ها و ذخیره‌ی آن‌ها در فایل‌های لازم نوشته شده است. قطعه کد ۱-۴ که در زیر آورده‌شده است، به تعداد تعیین‌شده گراف‌ها را از ورودی می‌خواند، و با اجرای مراحل توضیح داده شده، مدل های نظیر آن‌ها را تولید و ذخیره می‌کند.

```python
round=4
for i in range(round):
    path="C:/Users/Mrs/Desktop/prj/N/model"+str(i+1)+".mzn"
    mf = open(path, 'w')
    if i==round-1:
        break
    path="G"+str(i+1)+".txt"
    G1=read_dimacs_graph(path)
    n=nx.number_of_nodes(G1)
    d=dict(nx.all_pairs_shortest_path_length(G1))
    str_distance="|"

    for i in(range(n)):
        for j in(range(n)):
            str_distance+=str((d[str(i)][str(j)]))
            str_distance+=","
        str_distance=str_distance[0:len(str_distance)-1]
        str_distance+="|"

    mf.write("include \"globals.mzn\";\n")
    mf.write("/*variabes*/\n")
    mf.write("set of int: n=1.."+str(n)+";\n")
    mf.write("array[n] of var 1.."+str(n)+": T;\n")
    mf.write("var 2.."+str(n)+" :b;\n")
    mf.write("array[n,n] of var 0.."+str(n-1)+": D=["+str_distance+"];\n")

    mf.write("\n\n/*constraints*/\n")
    mf.write("constraint forall(j in n) ( exists(i in n where T[i]<=b) ( D[i,j]<=(b-T[i])));\n")
    mf.write("constraint alldifferent(T);\n")

    mf.write("\n")
    mf.write("solve minimize b;\n")
```

قطعه کد ۱-۴- خواندن گراف، تولید متغیرهای لازم و ایجاد و ذخیره‌ی مدل

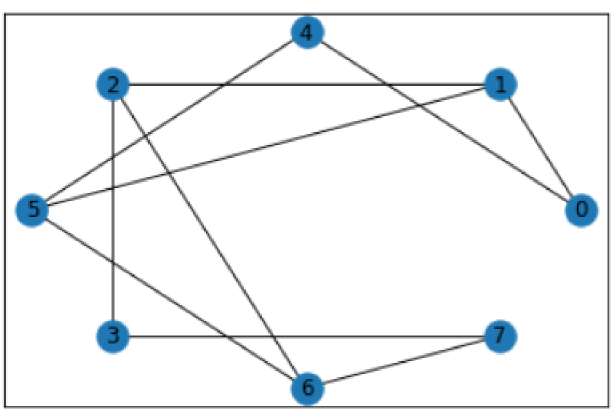
سایر بخش‌های کد از جمله توابع خواندن گراف‌ها و ... برای جلوگیری از طولانی شدن گزارش آورده نشده.

۲-۲-۴- اجرا در مینی زینک

پس از ایجاد مدل‌ها باید آن‌ها را اجرا، و حل‌ها را بررسی کرد. حال آنکه اجرای دستی تعداد زیادی مدل قطعاً سخت خواهد بود به همین دلیل از رابط میان مینی‌زینک و پایتون به منظور اجرای چندین مدل به طور پیاپی و ذخیره ی نتایج آن ها استفاده شده است. کد مذکور در ضمیمه ی گزارش موجود است، در این قسمت، به منظور نمایش یک نمونه حل با مدل بهینه‌سازی مقید، تنها یکی از مدل‌ها و راه‌حل مربوط به آن‌را در گزارش آورده‌ایم.

```
include "globals.mzn";
/*variabes*/
set of int: n=1..8;
array[n] of var 1..8: T;
var 2..8 :b;
array[n,n] of var 0..7: D=[|0,1,2,3,1,2,3,4|1,0,1,2,2,1,2,3|
2,1,0,1,3,2,1,2|3,2,1,0,4,3,2,1|1,2,3,4,0,1,2,3|
2,1,2,3,1,0,1,2|3,2,1,2,2,1,0,1|4,3,2,1,3,2,1,0|];


/*constraints*/
constraint forall(j in n) ( exists(i in n where T[i]<=b) (
D[i,j]<=(b-T[i])));
constraint alldifferent(T);

solve minimize b;
```

قطعه کد ۲-۴- مدل بهینه‌سازی مقید در مینی‌زینک



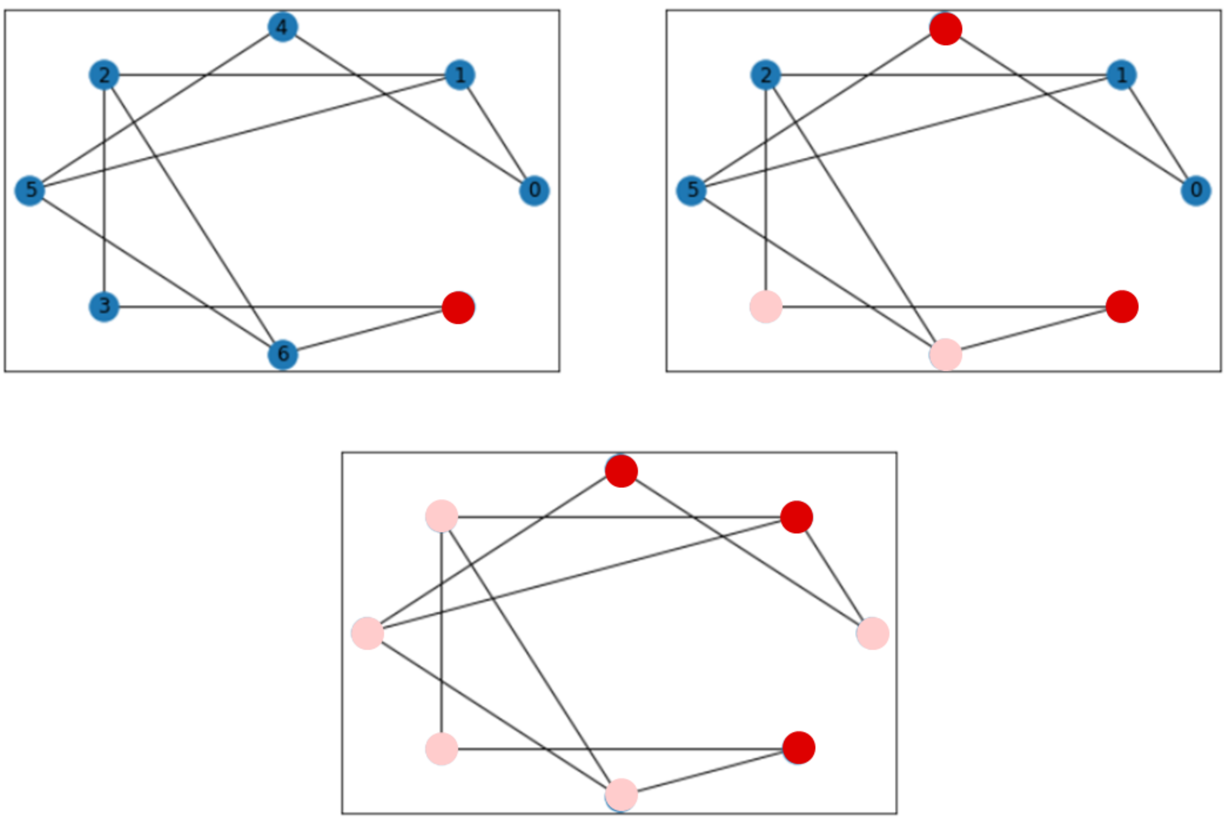
شکل ۱-۴- ورودی مدل بهینه سازی مقید

```
----------
T = array1d(1..8, [8, 3, 7, 6, 2, 5, 4, 1]);
b = 3;
----------
```

شکل ۲-۴- خروجی مدل بهینه‌سازی مقید

توجه شود که دنباله‌ی موجود در این خروجی ترتیب انتخاب نودها است و نه دنباله‌ی انتخاب، وجود عدد ۸ درابتدا به این معنی است که نود شماره‌ی ۰ در ۸ امین نوبت خواهد سوخت. بنابراین، ترتیب انتخاب نودها با توجه به خروجی به این شکل است:۲-۴

V7, V4, V1, …



شکل ۲-۴- نمایش راه حل حاصل از مدل بهینه‌سازی مقید

فصل پنجم: حل مسئله با برنامه‌ریزی خطی عدد صحیح

در کنار حل با یک مسئله‌ی بهینه‌سازی مقید، یکی دیگر از روش‌های مواجهه با مسئله، حل کردن آن با استفاده از برنامه‌ریزی خطی صحیح است که در آن ما سعی داریم مقدار بهینه‌ی یک تابع خطی را در عین ارضا کردن چند قید خطی روی متغیرهایی با مقادیر صحیح بیابیم. در این فصل ضمن بیان مسئله‌ی اشتعال گراف به شکل یک نمونه از مسئله‌ی برنامه‌ریزی خطی عدد صحیح، با استفاده از ابزارهای برنامه‌ریزی خطی عدد صحیح، آن را پیاده‌سازی می‌کنیم.

۵-۱- بیان اشتعال گراف به شکل برنامه‌ریزی خطی عدد صحیح

برنامه‌ریزی عدد خطی صحیح یک مسئله‌ی بهینه‌سازی ریاضیاتی است که در آن بعضی یا تمام متغیرها باید عدد صحیح باشند. در عین حال، تابع هدف و قیدها خطی هستند. این مسائل مشابه بهینه‌سازی خطی هستند و تنها تفاوت آن ها در وجود اجبار برای عدد صحیح بودن برخی از متغیرهاست. هدف برنامه‌ریزی خطی عدد صحیح یافتن مقدار بهینه‌ی تابع هدف خطی است. فضایی که در این مسائل با آن روبرو هستیم به دلیل وجود متغیرهای گسسته، نامحدب[۳۳] است و برخلاف برنامه‌ریزی خطی پیوسته، در زمان چندجمله‌ای قابل حل نیست.

[۳۳] Non-convex

۱-۱-۵- قیدها و متغیرها

حال که یک دید کلی به دست آوردیم، سعی می‌کنیم مسئله‌ی اشتعال گراف را به شکل یک برنامه‌ریزی خطی عدد صحیح بیان کنیم. متغیرها و قیدها را بیان کرده و درمورد هرکدام توضیحات لازم را ارائه می‌دهیم. توجه کنید که n و D پارامتر هستند و در ادامه توضیحات بیشتری درمورد آن‌ها خواهیم داد.

n

$D[i][j] \quad 1 \leq i,j \leq n$

$b \quad 2 \leq b \leq n$

$T[i] \quad 1 \leq i \leq n \quad 1 \leq T[i] \leq n$

$B[i][j] \quad 1 \leq i,j \leq n \quad 0 \leq B[i][j] \leq 1$

$a[i][k] \quad 1 \leq i,k \leq n \quad 0 \leq a[i][k] \leq 1$

$minimize\ b\ subject\ to:$

$$\sum_{j=1}^{n} B[i][j] > 0 \quad for\ all\ i$$

$$(2B[i][j]-1) * D[i][j] \leq b - T[j] \quad for\ all\ i,j$$

$$T[i] - \sum_{k=1}^{n} k * a[i][k] = 0 \quad for\ \ all\ i$$

$$\sum_{k=1}^{n} a[i][k] = 1 \quad for\ \ all\ i$$

$$\sum_{i=1}^{n} a[i][k] = 1 \quad for\ all\ k$$

در ادامه لازم است متغیرها، دامنه و قیدهای روی آن‌ها توضیح داده شود:

الف) متغیرها

- b معرف عدد اشتعال گراف است که قصد داریم آن را کمینه کنیم.

- T [i] یک آرایه به طول n است که در واقع نوبت انتخاب شدن نود i ام را بیان می‌کند و دقیقاً مشابه متغیر T [i] تعریف شده در فصل قبل است. برای یادآوری، می‌توان گفت در مسئله در هر مرحله علاوه بر اینکه تمام همسایه‌های نودهای از پیش سوخته شده می‌سوزند، یک نود جدید نیز برای سوختن انتخاب می‌شود و T [i] نوبت این انتخاب شدن را بیان می‌کند. مقادیری که T [i] اتخاذ می‌کند بین ۱ تا n خواهد بود، زیرا مقدار آن بیان گر یک نوبت برای سوختن است. برای مثال T[2]=3 بیان می‌دارد که نود شماره‌ی ۲ در سومین نوبت برای سوختن انتخاب خواهد شد.
- B [i] [j] یک متغیر باینری است که فقط مقادیر ۰ یا ۱ را اتخاذ می‌کند. اگر B [i] [j]=1 باشد به این مفهوم است که نود i توسط نود j سوزانده می‌شود. در صورتی که i≠j به این مفهوم است که j در یکی از مراحل سوخته است و این آتش طی چند مرحله به i رسیده و آن را سوزانده و اگر i=j باشد، یعنی نود به وسیله‌ی خودش سوزانده شده که به این مفهوم است که یکی از نودهای انتخاب شده برای سوختن بودن است.
- a [i] [k] یک متغیر باینری است که تنها مقادیر ۰ و۱ را اتخاذ می‌کند. این متغیر برای پیاده‌سازی تابع alldiff تعریف شده است. چون alldiff به خودی خود یک قید خطی نیست و هدف ما برنامه‌ریزی خطی صحیح است، می‌توان alldiff را طبق رویکرد [4] به وسیله‌ی چند قید خطی بیان کرد. در ادامه که قیدهای تعریف کننده‌ی alldiff را توضیح می‌دهیم درمورد این متغیر بیشتر توضیح خواهیم داد.

ب) پارامتر ها :

- n تعداد نودهای گراف است
- D [i] [j] فاصله‌ی بین نود i ام و j است و در واقع شامل کوتاه‌ترین مسیر بین دو به دوی نودها است.. برای مثال اگر D[2][3]=4 ، به این معناست که فاصله‌ی نود شماره‌ی ۲ و نود شماره‌ی ۳ در گراف برابر ۴ است.

متغیرهای a [i] [k] و B [i] [j] و T [i] و b قرار است پس از حل مدل مقداردهی شوند اما پارامترهای n و D [i] [j] مقادیری هستند که بر حسب گراف، باید محاسبه شوند و مقدار ثابت به آن‌ها داده می‌شود.

پ) قیدها

- سه قید آخر درواقع معادل alldiff هستند و درواقع alldiff(T [i]) را به طور خطی بیان می‌کنند. هدف این قیود این است که مقادیر T [i] متفاوت باشند، یعنی: $T[i] \neq T[j]\ iff\ i \neq j$ علت وجود این قید این است که T[i] نوبت انتخاب هر نود برای سوختن است و در هر نوبت فقط یک نود می‌تواند انتخاب شود پس ممکن نیست نوبت انتخاب دو نود برابر باشد. حال توضیح میدهیم که این قیدها چگونه این کار را می‌کنند.

- $T[i] - \sum_{k=1}^{n} k * a[i][k] = 0 \quad for \;\; all \; i$: این قید بیان میکند a[i][j]=1 در صورتی باید ۱ شود که T[i]=j، مثلا a[2][3]=1 یعنی T[2]=3 .
- $\sum_{k=1}^{n} a[i][k] = 1 \quad for \;\; all \; i$: این قید بیان میکند که هر مقدار، فقط به یک T[i] می‌تواند داده شود.
- $\sum_{i=1}^{n} a[i][k] = 1 \quad for \; all \; k$: این قید بیان میکند که به هر T[i] فقط یک مقدار می‌تواند نسبت داده شود.

- B [i] [j]=1 به معنای آن است که نود i با واسطه‌ی نود j سوزانده شده است، بنابراین می‌توان نتیجه گرفت قید $\sum_{j=1}^{n} B[i][j] > 0 \;\; for \; all \; i$ یعنی برای هر نود i باید حداقل یک نود وجود داشته باشد که به واسطه ی آن سوخته شده باشد.
- قید $(2B[i][j] - 1) * D[i][j] \le b - T[j] \quad for \; all \; i,j$ بیان می‌کند در انتهای b مرحله باید تمام نود ها سوخته باشند. حال بیایید توضیح دهیم که این قید چگونه این موضوع را تضمین میکند. برای این که راحت تر توضیحات را دنبال کنید، فراموش نکنید که وقتی b تعداد مراحل سوختن کل گراف است یعنی ما b مرحله فرصت داریم تا کل گراف را بسوزانیم.

  - اگر B [i] [j]=1 که یعنی نود i با واسطه‌ی نود j سوخته شده است، آنگاه حاصل عبارت $2B[i][j] - 1$ برابر با ۱ میشود، درنتیجه حاصل عبارت $(2B[i][j] - 1) * D[i][j]$ میشود $D[i][j]$ که برابر است با فاصله ی نود i از نود j و قید بیان میدارد، که این فاصله باید کمتر از $b - T[j]$ باشد، یعنی $D[i][j] \le b - T[i]$. در فصل قبل توضیح داده شد که الزام برقراری این نامساوی چیست. برای یادآوری میتوان گفت $b - T[i]$ به ما میگوید چند مرحله ی دیگر بعد از انتخاب i، تا انتهای مراحل (مرحله ی b ام) باقی مانده است و وقتی فاصله ی نود j با نود i کوچکتر یا مساوی تعداد مراحل باقی مانده باشد، آتش از i به j خواهد رسید. برای توضیحات بیشتر به بخش ۱-۱-۴ این گزارش مراجعه کنید.
  - اگر B [i] [j]=0 آنگاه حاصل عبارت $2B[i][j] - 1$ برابر با عدد ۱- میشود، درنتیجه حاصل عبارت $(2B[i][j] - 1) * D[i][j]$ یک عدد منفی خواهد بود و چون سمت راست نامساوی همواره نامنفی است، نامساوی همواره برقرار خواهد بود. علت طراحی آن است که وقتی B[i][j]=0 یعنی سوختن نود i ارتباطی به نود j ندارد پس لزومی ندارد $D[i][j] \le b - T[i]$ برقرار باشد.

ج) تابع هدف: minimize b بیان می‌کند که هدف مسئله کمینه کردن تعداد مراحل اشتعال است.

۲-۵- پیاده‌سازی

همانطور که ابزار های زیادی برای پیاده‌سازی مسائل بهینه‌سازی مقید وجود داشت، ابزارهای مفیدی نیز برای حل برنامه‌ریزی خطی عدد صحیح وجود دارد. یکی از این ابزار ها حل کننده ی پالپ[۳۴] است که در پایتون قابل استفاده است. در این بخش در ژوپیتر نوت بوک، با استفاده از حل کننده ی پالپ، مدل برنامه‌ریزی خطی عدد صحیح را پیاده کرده ایم.

۱-۲-۵- پیش پردازش

همانند فصل پیش، راه حل این فصل نیز احتیاج به پیش پردازشی جهت یافتن فاصله ی بین نود ها دارد. به این منظور از همان روش و کتابخانه های مذکور در فصل پیش استفاده شده است با این تفاوت که طبیعیتا قطعه کد تولید کننده ی مدل متفاوت است. چون پالپ به صورت کتابخوانه در پایتون مستقیما قابل استفاده است، نیازی به نوشتن مدل ها در فایل های جداگانه و سپس اجرای آن ها نبود و قابلیت اجرای هر مدل به محض پردازش گراف و یافتن فاصله ها و ساخت مدل، در همان محیط ژوپیتر نوت‌بوک وجود دارد.

```
G1=read_dimacs_graph(path)
n1=int(nx.number_of_nodes(G1))
d=dict(nx.all_pairs_shortest_path_length(G1))
D=[]
for i in(range(n1)):
    tmp=[]

    for j in(range(n1)):
        try:
            tmp.append(int((d[str(i)][str(j)])))
        except KeyError:
            tmp.append(n+1)

    D.append(tmp)
```

قطعه کد ۱-۵- پیش پردازش لازم برای پیاده‌سازی برنامه‌ریزی خطی عدد صحیح

[۳۴] PULP

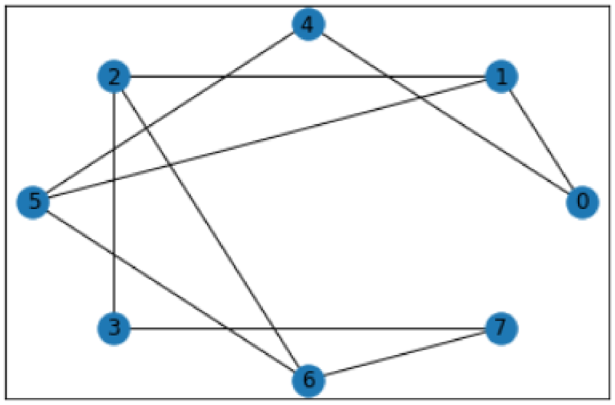
۲-۲-۵- پیاده‌سازی با پایتون

پس از پیش پردازش های لازم نوبت آن است که قید ها و متغیر های مسئله را تعریف کنیم و با امتحان کردن ورودی های مورد نظر راه حل آن ها را بررسی کنیم، قطعه کد زیر به منظور پیاده‌سازی قید ها و متغیر ها و هم چنین ذخیره ی راه حل های خروجی نوشته شده است.

```
burn = LpVariable("burn", 2, n1, cat=LpInteger)
T = pulp.LpVariable.dicts('T', n, lowBound=1,upBound=n1, cat=LpInteger)
B = pulp.LpVariable.dicts('B', (n,n) ,lowBound=0,upBound=1, cat=LpInteger)
a = pulp.LpVariable.dicts('a', (n,n) ,lowBound=0,upBound=1, cat=LpInteger)


prob += burn;
for i in n:
    for j in n:
        prob += ((num+1)*B[i][j]-num)*(D[i][j]+1)<=((burn-T[j])+1)

for i in n:
    prob+=lpSum([B[i][j]for j in n])>=1

for i in n:
    prob+=T[i]-lpSum([(k+1)*a[i][k] for k in n])==0

for i in n:
    prob+=lpSum([a[i][k] for k in n])==1

for k in n:
    prob+=lpSum([a[i][k] for i in n])==1
start_time = time.time()
status = prob.solve()
f.write('\n')
f.write(str(int(burn.value()))+" "+str((time.time() - start_time)))
```

قطعه کد ۲-۵- تعریف متغیرها و قیدهای برنامه‌ریزی خطی صحیح

در انتها نیز یک نمونه از ورودی و خروجی مربوط به آن را که در این مدل حل شده آورده ایم :



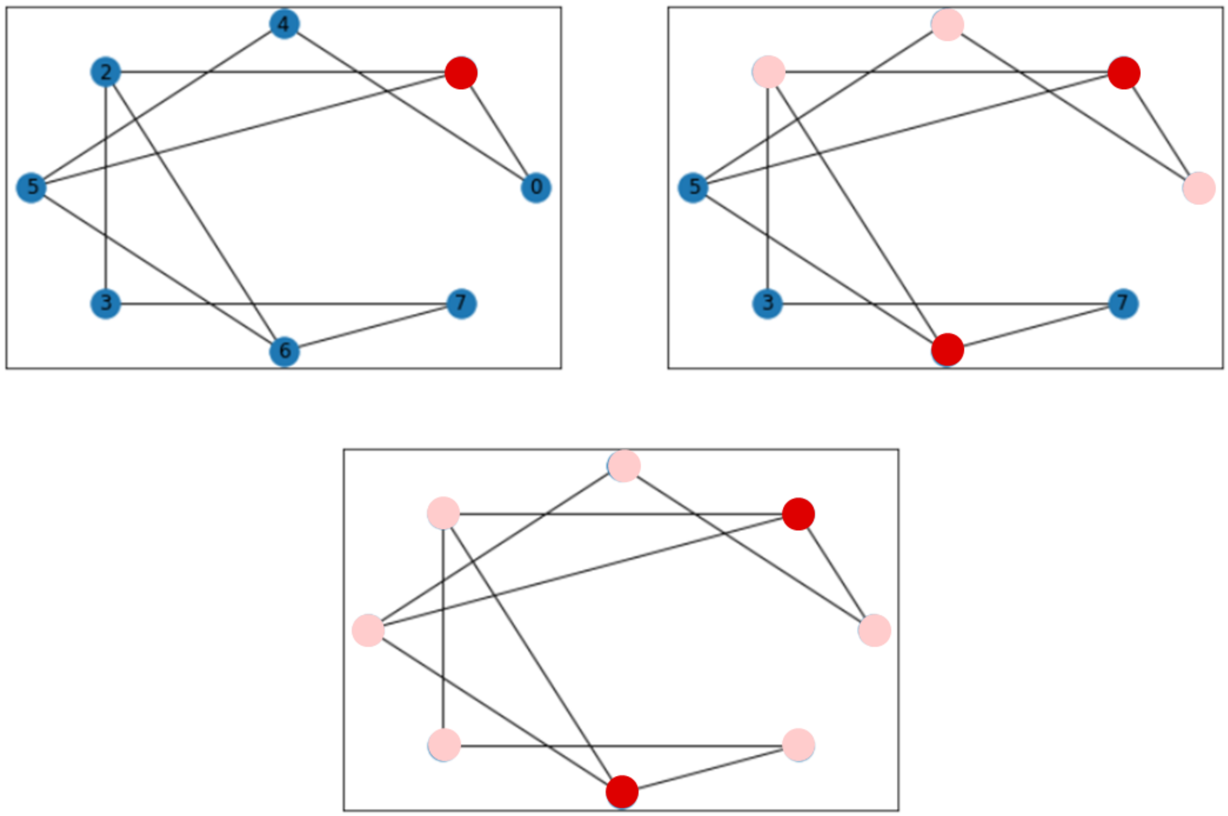
شکل ۱-۴- ورودی برنامه ریزی خطی عدد صحیح

```
[3.0, 1.0, 5.0, 6.0, 7.0, 8.0, 2.0, 4.0]
3.0
```

شکل ۲-۴- خروجی برنامه ریزی خطی عدد صحیح

در اینجا نیز دنباله ی خروجی، نماینده ی نوبت هاست یعنی نوبت سوختن نود ها به ترتیب زیر خواهد بود :

V1, V6, V0,. ...



شکل ۲-۴- نمایش راه حل حاصل از برنامه ریزی خطی صحیح

دیدیم با این که دنباله‌ی پیشنهادی توسط سه رویکرد عقب‌گرد، بهینه‌سازی مقید و برنامه‌ریزی خطی عددصحیح، متفاوت بودند اما همه ی آن ها عدد اشتغال یکسان و بهینه‌ی ۳ را معرفی می‌کنند.

فصل ششم: نتایج عددی

پس از معرفی و بررسی سه راه‌حل جستجوی عقب‌گرد، حل به وسیله‌ی مدل‌سازی با مسائل بهینه‌سازی مقید و حل به وسیله‌ی برنامه‌ریزی خطی عددصحیح، باید این راه‌حل‌ها را روی تعداد قابل قبولی گراف امتحان کرد. دو هدف اصلی از این کار وجود دارد، اول اطمینان از سازگار[۳۵] بودن راه حل ها با یکدیگر و دوم مقایسه‌ی راه حل ها.

۱-۷- انتخاب نمونه‌ها

برای اینکه بررسی و امتحان کردن راه حل ها بتواند دیدگاه مناسبی را در اختیار ما قرار دهد، لازم است نمونه‌هایی که برای تست استفاده می‌شود نمونه‌هایی باشد که به درستی انتخاب شده اند تا بتوانند نماینده‌ی حالات مختلفی که پیش خواهد آمد باشند. در طی پیاده سازی راه حل ها و تست‌های مختلفی که روی آن ها صورت گرفت، موارد مختلفی مشاهده شد که به عنوان یک دید اولیه برای انتخاب نمونه‌های اصلی تست می‌توانست مناسب باشد، با توجه به آن ها، موارد زیر در انتخاب نمونه‌های تست درنظر گرفته شدند:

۱) گراف کامل: چون گراف کامل به عنوان یک نمونه‌ی خاص در پیاده سازی راه‌حل عقب گرد مورد توجه قرار گرفته‌است، خوب است چند نمونه از این دسته از گراف‌ها با اندازه‌های متفاوت در تست حضور داشته باشند.

---

[۳۵] Consistent

۲) تنوع از لحاظ چگال[۳۶] یا خلوت[۳۷] بود گراف: در طی تست‌های اولیه دیده شد که چگال یا خلوت بودن گراف‌ها در سرعت راه حل ها بسیار موثر است از همین رو مناسب است این موضوع را در نمونه‌های تست نهایی مورد بررسی قرار دهیم و سپس درمورد نتایج آن بحث کنیم.

۳) تنوع از لحاظ تقارن گراف: علاوه بر موارد ذکر شده، تقارن گراف یکی از مواردی است که در حین تست‌های اولیه، به نظر می‌آمد در سرعت حل مسئله موثر باشد. به همین علت مناسب است این موضوع در نمونه‌هایی که در تست نهایی استفاده می‌شوند مورد توجه قرار گرفته گرفته و درمورد نتایج آن بحث شود.

۲-۷- نتایج

در این قسمت چند نمونه از مختلف از گراف‌ها را با توجه به معیارهای فوق مورد بررسی قرار می‌دهیم، در ابتدا باید بستر لازم برای تست این تعداد نمونه و ذخیره‌ی نتایج آن ها ایجاد شود و سپس خروجی آن ها با یکدیگر مقایسه شود.

۱-۲-۷- ایجاد بستر لازم

راه حل هایی که تا این مرحله پیاده سازی شده اند یک نمونه را دریافت می‌کنند و خروجی آن را نمایش می‌دهند، برای این که بتوان تعداد بیشتری نمونه را حل کرد و نتایج را ذخیره و سپس مقایسه کرد، لازم است تغییراتی در پیاده‌سازی‌ها ایجاد شود. این کار در دو مرحله صورت گرفته است:

۱) پیاده سازی عقب‌گرد از همان ابتدا طی یک حلقه نمونه‌های مختلفی را از ورودی خوانده و به ترتیب حل می‌کرد اما این موضوع در مورد پیاده‌سازی‌های دیگر صادق نبود. دلیل این موضوع این بود که دو راه حل دیگر مبتنی بر مدل‌سازی بودند و باید برای هر نمونه‌ی خاص یک مدل متفاوت نوشته می‌شد و سپس این مدل اجرا می‌شد پس لازم بود تغییراتی در پیاده‌سازی این راه حل ها صورت گیرد. برای مدل بهینه‌سازی مقید که در ابتدا فقط در مینی‌زینک پیاده سازی شده بود، به منظور ایجاد مدل‌های متفاوت برای هر گراف، یک کد پایتون نوشته شد و در ادامه نیز با استفاده از رابط میان مینی‌زینک و پایتون، در یک حلقه، تمام مدل‌ها به طور خودکار حل و راه‌حل آن‌ها ذخیره شد. در مورد مدل برنامه‌ریزی خطی عددصحیح نیز به همین شکل، کدی برای ایجاد مدل‌های گوناگون نظیر هر گراف به طور خودکار و حل آن ها در یک حلقه، نوشته شد

۲) راه حل های هر پیاده سازی در ابتدا صرفاً در خروجی نمایش داده می‌شد، برای اینکه بتوانیم راه حل های گوناگون را با یکدیگر مقایسه کنیم، لازم بود این راه حل ها در یک فرمت یکسان برای تمام راه حل ها ذخیره شود. فرمت پیاده شده به این شکل است که عدد اشتعال به دست آمده و مدت زمان حل آن نمونه‌ها، به شکل زیر در فایل‌های جداگانه برای هر راه حل، ذخیره می‌شود و سپس با استفاده از یک قطعه کد، سازگاری و مقایسه‌ی میانگین زمان این راه حل ها نسبت به یکدیگر انجام می‌شود.

---

[۳۶] Dense
[۳۷] Sparse

در زیر قطعه کد رابط پایتون و مینی‌زینک و همچنین فرمت خروجی راه حل ها را مشاهده می‌کنید.

```python
from minizinc import Instance, Model, Solver
import time

f = open("out_COP.txt", 'a')
for i in range(3):
    path="./model"+str(i+1)+".mzn"
    my_model = Model(path)
    gecode = Solver.lookup("gecode")
    instance = Instance(gecode, my_model)

    start_time = time.time()
    result = instance.solve()
    f.write('\n')
    f.write(str(result["b"])+" "+str((time.time() - start_time)))
```

قطعه کد ۱-۷- رابط پایتون و مینی‌زینک

```
4 0.10372257232666016
2 0.0
2 0.0009951591491699219
3 0.001993179321289062 5
3 0.0029935836791992188
3 0.0029916763305664062
3 0.0029916763305664062
3 0.001993894577026367
2 0.0010180473327636719
2 0.000991821289062 5
2 0.0005533695220947266
2 0.00045013427734375
2 0.0009982585906982422
2 0.0007517337799072266
2 0.0009796619415283203
2 0.0004904270172119141
2 0.0012631416320800781
2 0.0005650520324707031
2 0.0009980201721191406
```

شکل ۱-۷- فرمت خروجی راه حل ها

۲-۲-۷- خروجی‌ها

در این بخش، خروجی راه حل ها را روی انواع متنوعی از گراف مشاهده می‌کنیم. توجه کنید برای حل نمونه ها محدودیت زمانی ۵ دقیقه ای در نظر گرفته شده است و اگر حل بیش از موعد مقرر به طول بیانجامد، روند حل متوقف خواهد شد.

نوع نمونه	تعداد نود	عقب‌گرد		بهینه سازی مقید		برنامه ریزی خطی عدد صحیح	
		عدد اشتعال	زمان (ثانیه)	عدد اشتعال	زمان(ثانیه)	عدد اشتعال	زمان(ثانیه)
گراف کامل	5	2	0.0010180	2	0.6151707	2	0.0721704
	25	2	0.0009918	2	0.6813056	2	2.45166659
	45	2	0.0005566	2	0.8418848	2	7.14473462
	65	2	0.0004501	2	1.0503346	2	7.48140430
	85	2	0.0009982	2	1.23390221	2	26.64435172
	105	2	0.0007517	2	1.56257390	2	23.80230236
	125	2	0.0009796	2	1.87365269	-	LIMIT EXCEEDED
	145	2	0.0004904	2	2.68635988	-	LIMIT EXCEEDED
	165	2	0.0012631	2	2.67342805	-	LIMIT EXCEEDED
	185	2	0.0005650	2	3.15221858	-	LIMIT EXCEEDED

جدول ۱-۷- خروجی راه حل ها روی چند نمونه گراف کامل

نوع نمونه	تعداد نود	تعداد یال	عقب‌گرد		بهینه سازی مقید		برنامه ریزی خطی عدد صحیح	
			عدد اشتعال	زمان (ثانیه)	عدد اشتعال	زمان(ثانیه)	عدد اشتعال	زمان(ثانیه)
متقارن	50	1000	3	0.4606399	3	19.0034236	3	72.029413
	50	500	4	2.3465876	4	152.3100124	4	212.12034
	50	200	3	6.7947790	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
نامتقارن	50	1000	3	0.5900041	3	20.1334500	3	58.524763
	50	500	3	2.0009843	3	138.401234	-	LIMIT EXCEEDED
	50	200	3	4.0823532	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED

جدول ۲-۷- خروجی راه حل ها روی چند نمونه گراف با ۵۰ نود

نوع نمونه	تعداد نود	تعداد یال	عقب‌گرد		بهینه سازی مقید		برنامه ریزی خطی عدد صحیح	
			عدد اشتعال	زمان (ثانیه)	عدد اشتعال	زمان(ثانیه)	عدد اشتعال	زمان(ثانیه)
متقارن	100	750	3	83.614463	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	100	2000	3	21.650700	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	100	4000	3	5.7855477	3	197.168405	-	LIMIT EXCEEDED
نامتقارن	100	750	4	69.9978135	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	100	2000	3	14.5497067	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	100	4000	3	1.0001291	3	184.9426052	-	LIMIT EXCEEDED

جدول ۳-۷- خروجی راه حل ها روی چند نمونه گراف با ۱۰۰ نود

نوع نمونه	تعداد نود	تعداد یال	عقب‌گرد		بهینه سازی مقید		برنامه ریزی خطی عدد صحیح	
			عدد اشتعال	زمان (ثانیه)	عدد اشتعال	زمان(ثانیه)	عدد اشتعال	زمان(ثانیه)
متقارن	150	500	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	150	5000	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	150	11000	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
نامتقارن	150	500	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	150	5000	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED
	150	11000	4	253.1540091	-	LIMIT EXCEEDED	-	LIMIT EXCEEDED

جدول ۴-۷- خروجی راه حل ها روی چند نمونه گراف با ۱۵۰ نود

۳-۷- مقایسه‌ی نتایج

به طور کلی، نتیجه‌ی حاصل از تست داده‌ها در سه راه حل دقیق را می‌توان به صورت زیر جمعع بندی نمود:

- طبق داده‌های تست شده هر سه راه حل با یکدیگر سازگار هستند و تناقضی در پاسخ‌های آن ها وجود ندارد.
- برای گراف‌هایی که تقارن کمی دارند، راه‌حل‌های دقیق بسیار خوب عمل می‌کنند زیرا انجام مراحل سوزاندن با شروع‌های مختلف، نتایج بسیار متفاوتی را ارائه می‌دهد و در نتیجه شاخه‌های زیادی در حین جستجو هرس خواهد شد.
- راه‌حل‌ها برای گراف‌هایی که سنگین هستند، یا به عبارت دیگر تعداد یال‌های زیادی دارند، بهتر عمل می‌کنند زیرا نودهای زیادی در هر مرحله به واسطه‌ی همسایگی با سایر نودها می‌سوزد و در نتیجه گراف عدد اشتعال پایینی داشته و حل آن زمان بر نخواهد بود.
- برای نمونه‌های بزرگ، به خصوص اگر متقارن و نسبتاً خلوت باشند، راه حل های دقیق زمان غیرقابل قبولی را نیاز دارند بنابراین بهتر است در این موارد از راه حل غیر دقیق استفاده شود.

فصل هفتم: حل مسئله با الگوریتم ژنتیک

تا این جا به بررسی و پیاده‌سازی الگوریتم‌های دقیق برای حل مسئله‌ی اشتعال گراف پرداختیم. راه حل های دقیق هرچند پاسخ بهینه را برمی‌گردانند اما ممکن است در شرایط مختلف، مثلاً برای یک نمونه ورودی بزرگ یا پیچیده، زمان و حافظه‌ی بسیار زیادی نیاز داشته باشند که ما در عمل به آن دسترسی نداشته باشیم، بنابراین داشتن یک راه حل تقریبی برای این موارد می‌تواند مفید باشد. در این فصل به تعریف مسئله‌ی اشتعال گراف در قالب یک الگوریتم ژنتیک، به عنوان یک روش رایج حل تقریبی مسائل می‌پردازیم.

۱-۶- مدل ژنتیک مسئله

الگوریتم ژنتیک یک روش ابتکاری حل مسائل در علوم کامپیوتر است که با الهام گرفتن از رویکردهای طبیعی مثل جهش، انتخاب طبیعی و ... سعی در حل مسائل دارد. این الگوریتم در حل مسائل بهینه‌سازی و جستجو، اغلب به راه حل های بسیار خوبی می‌رسد.

بخش‌های اصلی یک مدل برای الگوریتم ژنتیک عبارت‌اند از:

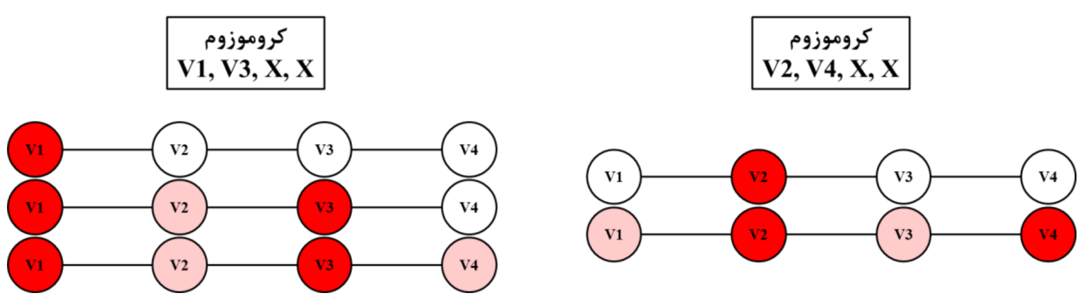
- کروموزوم[۳۸]
- تابع تقطیع[۳۹]
- تابع برازندگی[۴۰]
- تابع جهش[۴۱]

با این تصور که مخاطب این گزارش با مفاهیم فوق آشنایی دارد، در ادامه یک توضیح اجمالی در مورد هر مفهوم و پیاده‌سازی مسئله‌ی اشتعال گراف با الگوریتم ژنتیک را مورد بحث قرار خواهیم داد.

۱-۱-۶- کروموزوم

هر کروموزوم درواقع بیانگر یک وضعیت[۴۲] از مسئله است که بنا بر مسئله تعریف می‌شود. ما در تلاش خواهیم بود وضعیت مسئله را بهبود ببخشیم تا به وضعیت بهینه که راه حل مطلوب ما است نزدیک شویم. در مسئله‌ی اشتعال گراف هر وضعیت از مسئله در واقع یک ترتیب انتخاب نود است که بهتر بودن یک وضعیت نسبت به دیگری درواقع عدد اشتعال کمتر حاصل از ترتیب انتخابی سوختن است.

طبق توضیحات داده شده، یک نمایش مناسب برای مسئله‌ی اشتعال گراف، می‌تواند جایگشتی از شماره‌ی متناظر نودها باشد که بیانگر ترتیب انتخاب آن ها خواهد بود.



شکل ۱-۶- راه حل های نظیر دو کروموزوم متفاوت برای مسئله

در تصویر بالا دو راه حل مختلف برای سوزاندن گراف مسیر با ۴ نود را میبینیم که یکی از آن ها بهینه است و دیگری خیر. میتوان دید هرکدام از این راه حل ها قابل تطابق با یک جایگشت انتخابی از سوختن نود ها هستند. باید توجه شود که بعد از چند مرحله انتخاب نود، تمام نود ها خواهد سوخت، برای مثال در نمونه ی سمت راست پس از انتخاب های

---

[۳۸] Chromosome
[۳۹] Crossover function
[۴۰] Fitness function
[۴۱] Mutation function
[۴۲] state

V2 و V4 تمام نود ها میسوزند پس عملا مهم نیست که نود های بعدی در جایگشت چه هستند و تاثیری در میزان کیفیت راه حل ندارند پس آن ها را با X مشخص کرده ایم.

نکته ی دیگر این که نودی که در هر مرحله انتخاب میشود باید از میان نود های سوخته نشده باشد. در قسمت تعریف تابع برازندگی، این موضوع را بررسی خواهیم کرد.

۲-۱-۶- تابع تقطیع

تابع تقطیع تابعی است که ورودی آن دو کروموزوم و خروجی آن یک یا چند کروموزوم متفاوت است که به روشی از روی کروموزوم‌های والد ساخته شده. هدف از این کار ساختن کروموزوم‌های جدید به منظور ملاقات وضعیت‌های جدید مسئله است. کروموزوم‌های تعریف شده برای مسئله‌ی ما به صورت دنباله‌ای از اعداد هستند و روش‌های زیادی برای ساخت یک دنباله‌ی جدید از روی دو دنباله‌ی والد وجود دارد، اما نکته‌ی قابل توجه این است که در این مسئله هر دنباله‌ی حاصل، قابل قبول نخواهد بود. دنباله‌های نظیر کروموزوم‌های ما همان‌طور که توضیح داده شد باید به شکل جایگشت باشند به همین دلیل باید به دنبال توابع تقطیعی باشیم که این قاعده را حفظ کنند. توابع تقطیع ترتیبی[43] این قاعده را حفظ می‌کنند.

انواع زیادی از توابع تقطیع ترتیبی وجود دارد، یکی از آن ها روش تقطیع ترتیبی یکنواخت[44] است، با توجه به [5] در این روش که ورودی و خروجی آن دنباله‌هایی به شکل جایگشت هستند، یک دنباله ی باینری از ۰ و ۱ به طول کروموزوم، به طور تصادفی و با استفاده از تابع توزیع یکنواخت[45] تولید میکنیم، سپس ژن های نظیر ۱ها در دنباله ی تولید شده را عینا از والد اول کپی می‌کنیم و سپس ژن هایی که از والد اول انتخاب نشده اند را از والد دوم، به همان ترتیب، در نتیجه قرار می‌دهیم.

1	2	3	4	5	6	7	8
1	0	0	1	1	0	1	0
8	3	7	2	4	1	5	6
1	8	3	4	5	2	7	6

شکل ۲-۶- نمونه از از تقطیع ترتیبی یکنواخت

[43] Ordered crossover
[44] Uniform ordered crossover
[45] Uniform distribution function

۳-۱-۶- تابع برازندگی

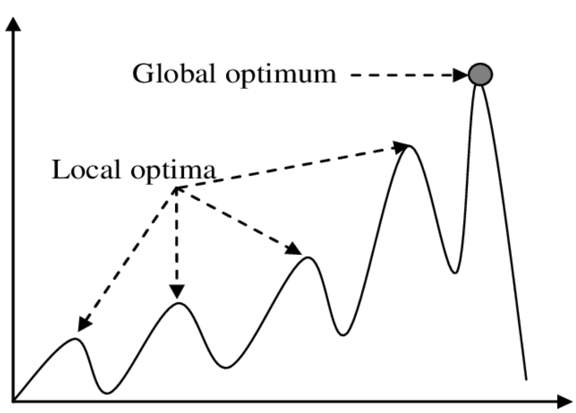
همان‌طور که گفته شد، ما به دنبال یافتن وضعیت بهینه برای مسئله هستیم پس باید روشی برای مقایسه‌ی استیت‌ها با یکدیگر باشیم. تابع برازندگی یک استیت را دریافت می‌کند و ارزش آن را تعیین می‌کند. در مسائل بهینه‌سازی اصولاً تابع هدف، گزینه‌ای است که برای استفاده به عنوان تابع برازندگی انتخاب می‌شود. در مورد مسئله‌ی مورد بحث ما نیز تابع برازندگی، تابعی است که عدد اشتعال متناظر با دنباله‌ی انتخاب را محاسبه می‌کند. هرچه عدد اشتعال متناظر با یک وضعیت کمتر باشد طبیعتاً آن وضعیت بهتر است.

نکته ی قابل توجه این است که نودی که در هر مرحله انتخاب میشود باید از میان نود های سوخته نشده باشد، پس اگر جایگشتی این قاعده را نقض کند، نباید شانس رفتن به نسل های بعدی را داشته باشد. به همین منظور در محاسبه ی تابع برازندگی برای نقض این قانون یک جریمه‌ی قابل توجه قرار داده می‌شود.

پس به عنوان جمع بندی میتوان گفت تابع برازندگی برای این مسئله مراحل سوختن نظیر دنباله را روی گراف اجرا می‌کند و عدد اشتعال نظیر آن را برمی‌گرداند و اگر دردنباله، از قانون گفته شده تخطی شده باشد برای آن جریمه در نظر گرفته میشود. میتوان برای پیاده‌سازی این تابع برازندگی، از توابعی که برای محاسبه ی عدد اشتعال در پیاده‌سازی راه حل های دقیق دیدیم، استفاده کرد.

۴-۱-۶- تابع جهش

وقتی در قسمتی از فضای حالت مدام به سمت وضعیت‌های بهتر آن قسمت حرکت کنیم، خطر گیرافتادن در بهینه‌ی محلی[۴۶] وجود دارد. منظور از بهینه‌ی محلی نقطه‌ای از فضای حالت است که در مقیاس بخش‌های اطراف خود بهینه است اما در سرتاسر فضای حالت بهینه نیست. طبیعتاً گیرافتادن در بهینه‌ی محلی در حالی که در تلاش برای حل یک مسئله‌ی بهینه‌سازی سراسری[۴۷] هستیم، خوشایند نیست. یک تکنیک برای خارج شدن از بهینه‌های محلی استفاده از جهش است.



شکل ۳-۶- تفاوت بهینه ی محلی و سراسری

[۴۶] Local optima
[۴۷] Global optimization

منظور از جهش ایجاد تغییرات به طور تصادفی در کروموزوم‌های حاصل از تقطیع است. جهش باعث می‌شود در کروموزوم‌ها تنوع ایجاد شود و کروموزوم‌های در دسترس مربوط به وضعیت‌هایی در قسمت‌های متنوعی از فضای حالت باشند. این امر خطر گیر کردن در بهینه‌ی محلی را کاهش می‌دهد.

باید توجه داشت که تابع جهش می‌بایست فرمت قابل قبول کروموزوم‌ها را حفظ کند. در مسئله‌ی ما کروموزوم‌ها به شکل جایگشت نودها هستند بنابراین باید توجه داشته باشیم که جهش این فرمت جایگشتی را حفظ کند. یک تابع مناسب برای این هدف، جابه‌جا کردن دو نود تصادفی از جایگشت است.

فصل هشتم: نتیجه‌گیری

در این گزارش، سعی شد دید خوبی نسبت به مسئله‌ی اشتعال گراف به خواننده داده شود و با معرفی آن و آوردن مثال‌های کافی در فصل دوم، ماهیت و اهمیت مسئله مشخص شود. سپس مسئله را با سه رویکرد دقیق و یک رویکرد ابتکاری غیر دقیق مورد بحث و بررسی قرار داده و آن را حل کردیم.

رویکردهای دقیق شامل حل به‌وسیله‌ی جستجوی عقب‌گرد، حل با مسائل بهینه‌سازی مقید و حل با برنامه‌ریزی خطی عدد صحیح بود. در فصل سوم به طور گسترده به پیاده‌سازی و بهبود جستجوی عقب‌گرد پرداختیم و توانستیم جستجو را نسبت به حالت عادی، بهبود دهیم و از هیوریستیک ها و کران هایی استفاده کنیم که فضای مسئله را کاهش می‌دهند. در فصل چهارم و پنجم مسئله را به وسیله ی قیدها و متغیرها بیان کردیم و با ابزار های در درسترس راه حل را پیاده کردیم و مشاهده کردیم مدل مطرح شده برای مسئله چه در حالت بهینه‌سازی مقید و چه در حالت برنامه‌ریزی خطی عدد صحیح، درست کار می‌کند. هم‌چنین نتایج حاصل از سه روش دقیق با یکدیگر سازگار بود.

در نهایت برای حالاتی که مسئله ممکن است فضا و زمان زیادی را برای حل نیاز داشته باشد، یک راه حل غیردقیق ارائه دادیم و موفق شدیم با بیان مسئله به روش الگوریتم ژنتیک یک راه حل غیردقیق برای آن ارائه دهیم.

هم چنین دیدیم، راه حل های ارائه شده روی نمونه های تست شده رفتار های زیر را داشتند :

- هر سه راه حل با یکدیگر سازگار بودند.
- راه حل ها روی گراف های نامتقارن بهتر عمل میکنند.
- راه حل ها روی گراف های چگال بهتر عمل میکنند
- برای نمونه های بزرگ تر خصوصا اگر نامتقارن باشند بهتر است از راه حل ها غیردقیق استفاده شود.

در انتها، باید گفت در این پروژه برای حل مسئله‌ی اشتعال گراف گام‌های قابل توجهی برداشته شد و می‌توان گفت راه‌حل‌های قابل قبولی برای حل دقیق این مسئله پدید آمد. باید در نظر داشت که حل این مسئله به صورت غیر دقیق هنوز جای کار دارد و می‌توان با استفاده از سایر الگوریتم‌های جستجوی محلی و متاهیوریستیک[۴۸] مانند تبرید شبیه سازی شده[۴۹] و یا با استفاده از تکنیک‌های یادگیری ماشین[۵۰] و یادگیری عمیق[۵۱] به حل آن پرداخت.

[۴۸] Metaheuristic
[۴۹] Simulated annealing
[۵۰] Machine learning
[۵۱] Deep learning

مراجع

[1] J. Janssen, A. Bonato and E. Roshanbin, "Burning a graph as a model of social contagion," 2014.

[2] Z. Rezai Farokh, M. Tahmasebi, Z. Haj Rajab Ali Tehrani and Y. Buali, "New heuristics for burning graphs," 2020.

[3] A. Bonato, "A survey of graph burning," 2020.

[4] H. Yan, "Representations of the all_different Predicate of Constraint Satisfaction in Integer Programming," 2001.

[5] G. Syswerda, "Uniform Crossover in Genetic Algorithms," 1989.