دانشکده علوم ریاضی

سمینار درس بهینه سازی ترکیبیاتی

مسأله رنگ آمیزی گراف

استاد ارجمند

آقای دکتر رضا قنبری

نگارش

سیده عاطفه موسوی اصل

زمستان ۱۳۹۰

فهرست مطالب

فصل ۱

۱.۱ مقدمه

تعداد زیادی از مسائل کاربردی مانند مسائل برنامه ریزی و مسائل ذخیره سازی هستند که در آن ها لازم است مجموعه رأسی گراف مربوط را افراز کنیم به گونه ای که رأس های مجاور، به مجموعه های متفاوتی از افراز تعلق داشته باشند. برای توضیح بیشتر، به این مثال توجه کنید. فرض کنید، تعدادی ماده شیمیایی باید انبار و ذخیره شوند. جفت های خاص از این مواد شیمیایی را نمی توان در ظرف یکسانی ذخیره کرد، چون با یکدیگر واکنش انجام می دهند. کمترین تعداد از ظروف مورد نیاز برای ذخیره این مواد شیمیایی به گونه ای که هیچ دو ماده شیمیایی که با هم واکنش انجام می دهند در ظرف یکسانی قرار نگیرند، چیست؟ می توانیم این حالت را با گراف G که رأس های آن متناظر با مواد

شیمیایی است، مدلسازی کنیم. دو رأس با یك یال به هم وصل شده اند اگر مواد شیمیایی متناظر با هم واکنش انجام ندهند. بنابراین، مواد شیمیایی در یك ظرف، متناظر با یك مجموعه مستقل از رأس ها در G است. در واقع مسأله رنگ آمیزی گراف بدون جهت $G(V,E)$ مسأله پیدا کردن کمترین تعداد رنگ برای اختصاص دادن به رئوس G است طوری که رئوس مجاور همرنگ نباشند. در این جا برآنیم تا روش هایی برای حل این مسأله ارائه دهیم.

۲.۱ تعاریف اولیه

ابتدا برخی تعاریف مربوط به رنگ آمیزی گراف را بیان می کنیم:

۱.۲.۱ رنگ آمیزی رأسی

فرض کنید $G(V,E)$ یک گراف ساده باشد. یک رنگ آمیزی رأسی[۱] G یک نگاشت به صورت $f : V(G) \longrightarrow N$ با این شرط است که اگر $uv \in E(G)$ ، داشته باشیم: $f(u) \neq f(v)$. عدد $f(v)$ رنگ رأس v نامیده می شود.

[۱] vertex coloring

۲.۲.۱ k-رنگ پذیری

در صورتی که امکان رنگ آمیزی رئوس گراف با kرنگ ، وجود داشته باشد، گراف را k-رنگ پذیر[۱] رأسی می نامند.

۳.۲.۱ عدد رنگی

کمترین تعداد رنگ لازم برای رنگ آمیزی گراف G ، عدد رنگی[۲] گراف G ، نامیده می شود و با $\chi(G)$ نشان داده می شود. به عبارت دیگر $\chi(G)$ کمترین kای است که به ازای آن یك گراف kبخشی است.

گراف G با عدد رنگی k، k-رنگی نیز نامیده می شود.

۴.۲.۱ رده رنگی

مجموعه رئوسی از گراف G را که پس از رنگ آمیزی، رنگ یکسانی به آن ها اختصاص داده می شود، یك رده رنگی[۳] نامیده می شود. اگر رنگ آمیزی موردنظر یك kرنگ آمیزی باشد، آن گاه k رده رنگی وجود دارد. درواقع مجموعه رأس های همرنگ (رده های رنگی)، یک مجموعه پایدار، یا یک زیرمجموعه مستقل از $V(G)$ را تشکیل می دهند، به

[۱] k-colorable [۲] chromatic number [۳] color class

این معنی که بین هر دو رأس که عضو یك رده رنگی هستند، یالی موجود نیست.

۵.۲.۱ مسأله رنگ آمیزی گراف

فرض کنید گراف G و عدد k داده شده اند. آیا می توان گراف G را با k رنگ، رنگ آمیزی رأسی کرد؟ این مسأله تصمیم گیری رنگ آمیزی رأسی است و مسأله بهینه سازی آن عبارت است از یافتن کمترین تعداد رنگ لازم برای رنگ آمیزی رأسی گراف G. به عبارت دیگر تعیین عدد رنگی گراف G.

مسأله رنگ آمیزی گراف[۱]، یك مسأله به طور قوی Np-hard [۲] است. در نتیجه در حالت کلی، الگوریتمی که در زمان چندجمله ای بتواند این مسأله را به طور دقیق حل کند وجود ندارد. برای اثبات این موضوع، کافی است حالت خاص مسأله ۳-رنگ پذیری را در نظر گرفته و آن را به مسأله ۳-SAT کاهش دهیم. [۱۳] ، [۱۴] ،[۳۱]

رنگ آمیزی گراف بیشتر در مورد مسائلی کاربرد پیدا می کند که در آن ها بایستی از مجاورت دو عنصر جلوگیری شود. در هر کاربرد، عناصر و مجاور بودن نسبت به همان مسأله تعریف می شود و در نتیجه رئوس و یال های گراف، بدست می آیند و با رنگ آمیزی این گراف جواب مسأله حاصل می شود. به عنوان مثال، در مسائل زمان بندی [۳]

[۱] Graph Coloring Problem [۲] Strongly Np-hard [۳] Timetabling

، جدول سودوکو [۱]، اختصاص فرکانس به ایستگاه های رادیویی[۲] ،مکان یابی ثبت[۳]، و بسیاری از مسائل دیگر به کاربرد رنگ آمیزی گراف برمی خوریم.

۳.۱ مدل برنامه ریزی خطی عدد صحیح

مسأله رنگ آمیزی گراف را می توان به صورت برنامه ریزی عدد صحیح مدل بندی کرد.

فرض کنید $i \in V$ و $۱ \leq j \leq n$. متغیر x_{ij} را به صورت زیر در نظر می گیریم:

$$x_{ij} = \begin{cases} ۱ & \text{اگر رنگ j به رأس i اختصاص داده شده باشد} \\ ۰ & \text{در غیر اینصورت} \end{cases}$$

همچنین n متغیر w_j را به صورت زیر تعریف می کنیم:

$$w_j = \begin{cases} ۱ & \text{اگر رنگ j در رنگ آمیزی رأسی به کار رفته باشد} \\ ۰ & \text{در غیر اینصورت} \end{cases}$$

به عبارت دیگر، $w_j = ۱$ اگر رأس i وجود داشته باشد طوری که $x_{ij} = ۱$. حال مدل برنامه ریزی عدد صحیح را بر مبنای متغیرهای ذکر شده ارائه می دهیم:

---

[۱] Sudoku [۲] Mobile radio frequency assignment [۳] Register allocation

$$\min \quad \sum_{j=۱}^{n} w_j$$
$$subject\ to: \begin{cases} \sum_{j=۱}^{n} x_{ij} = ۱ & \forall i \in V \\ x_{ij} + x_{kj} \leq w_j \quad \forall \{i,k\} \in E, ۱ \leq j \leq n \\ x_{ij} \in \{۰, ۱\} \forall i \in V, ۱ \leq j \leq n \end{cases} \tag{۱.۱}$$

۴.۱ تاریخچه رنگ آمیزی

بحث رنگ آمیزی گراف ها با ارائه حدس چهار رنگ آغاز شد. حدس چهار رنگ این سوال را می پرسد که آیا می توان یک گراف مسطح را با چهار رنگ، رنگ آمیزی کرد طوری که رئوس مجاور همرنگ نباشند؟ این حدس، اولین بار توسط فرانسیس گاتری[۱] مطرح شد. او ابتدا در دانشگاه لندن دانشجوی آگوستس دمورگان[۲] بود و پس از فارغ التحصیل شدن شروع به تحصیل در رشته حقوق نمود. اما چند سال بعد به برادرش فردریك گاتری [۳]که دانشجوی دمورگان شده بود، نتایج تلاش هایی را که در زمینه رنگ آمیزی نقشه ها انجام داده بود، نشان داد و از او خواست که از استادش در مورد آن ها سؤال کند. دمورگان نتوانست جوابی به او بدهد، اما در تاریخ ۲۳ اکتبر ۱۸۵۲، یعنی در همان تاریخی که این سوال از او پرسیده شد، نامه ای به ویلیام همیلتن [۴] نوشت و سوالی

[۱] Francis Guthrie [۲] Augusts De Morgan [۳] Frederick Guthrie [۴] William Hamilton

را که از او پرسیده شده بود، برای او توضیح داد. همیلتن در تاریخ ۲۶ اکتبر ۱۸۵۲ به نامه او پاسخ داد و بیان کرد که به زودی نمی تواند به سوال او پاسخ دهد.

اولین نتایج توسط آرتور کیلی[۱]در مقاله ای منتشر شد[۱]. در ۱۷ جولای ۱۸۷۹، آلفرد بری کمپ [۲] اعلام کرد که یک اثبات برای حدس چهار رنگ بدست آورده است. با پیشنهاد کیلی او قضیه را به ژورنال ریاضیات در آمریکا ارائه کرد، که در اواخر سال ۱۸۷۹ به چاپ رسید.

قضیه چهار رنگ در سال ۱۸۹۰ مجددا به حدس چهار رنگ تبدیل شد! هیوود [۳] که یک مدرس در انگلستان بود، مقاله ای با عنوان "قضیه رنگ آمیزی نقشه" منتشر کرد و در آن هدف خود را بیان کرد. هیوود در مقاله اش، علاوه بر این که نشان داد اثبات کمپ اشتباه است، اثبات کرد که هر نقشه ۵-رنگ پذیر است.

این وضعیت تا سال ۱۹۷۶ ادامه داشت تا اینکه بالاخره حدس چهار رنگ توسط اپل [۴]و هکن [۵]به اثبات رسید و آن ها در بعضی کارهای الگوریتمی از جان کوك [۶] کمک گرفتند. [۲۰] ، [۲۱]

آخرین تلاش ها در سال ۱۹۹۶ با ارائه اثبات جدیدی برای قضیه چهار رنگ توسط ربرتسون [۷] ، سندرس [۸]، سیمور [۹]و توماس [۱۰] انجام شده است. [۲۶] ، [۲۷]

---

[۱]Arthur Cayley [۲]Alfred Bray Kempe [۳]Percy John Heawood [۴]Kenneth Appel

[۵]Wolfgang Haken [۶]John Koch [۷]Robertson [۸]Sanders [۹]Seymour [۱۰]Thomas

فصل ۲

روش های حل مسأله رنگ آمیزی

همان طور که بیان شد، مسأله رنگ آمیزی گراف، در صورت دلخواه بودن گراف مربوطه، یك مسأله Np-hard است، اما برای گروه های خاصی از گراف ها، مانند گراف های تام[۱]، در زمان چندجمله ای قابل حل دقیق می باشد. یکی از مشهورترین الگوریتم های دقیق برای حل مسأله رنگ آمیزی، الگوریتمی ساده و کارا به نام DSATUR است. [۱۰] در واقع این الگوریتم، یك الگوریتم شاخه و کران است که از شمارش ضمنی استفاده می کند و تا کنون روش های متعددی برای بهبود آن ارائه شده است.[۲۹] علاوه بر آن، روش هایی مانند الگوریتم تولید ستون [۴]، الگوریتم صفحه برش [۱۷]، الگوریتم های شاخه

[۱] perfect graphs

و برش [۳۰] و بسیاری الگوریتم های دیگر در سال های اخیر ارائه شده اند که این مسأله را به طور دقیق حل می کنند اما نتایج تجربی نشان داده اند که این الگوریتم ها اکثراً برای گراف هایی خوب عمل می کنند که حداکثر ۱۰۰ رأس داشته باشند و این تعداد رأس، در مقایسه با مسائل واقعی بسیار کم است.

عده زیادی تلاش کردند تا مسأله رنگ آمیزی گراف را با روش های جستجوی فضا حل کنند . به عنوان مثال در [۲۴]، [۲۳] و [۱۹] روش هایی برای یافتن عدد رنگی از طریق جستجوی محلی فضا ارائه شده است.

از طرفی، به دلیل Np-hard بودن مسأله رنگ آمیزی گراف، دانشمندان زیادی علاقه مند شده اند تا توسط الگوریتم های ابتکاری [۱] و فراابتکاری [۲] جواب های قابل قبول تری برای آن ارائه دهند. به عنوان مثال الگوریتم های دنباله ای مانند الگوریتم حریصانه [۳] [۶]، یا الگوریتم های فراابتکاری مانند سرد کردن تدریجی [۴] [۲۵]، جستجوی ممنوع [۵] [۲] ، جستجوی همسایگی متغیر [۸] [۶] ، جستجوی فضای متغیر [۷] [۳]، مورچگان [۸] [۳۲] ، ژنتیك [۹] [۹] ، الگوریتم حافظه تطبیقی [۱۰] [۲۸] و بعضی الگوریتم های دیگر ارئه شده اند.

---

[۱]heuristics [۲]metaheuristics [۳]Greedy Graph Coloring [۴]Simulated Annealing [۵]Tabu Search [۶]Variable Neighborhood Search [۷]Variable Space Search [۸]Ant Colony [۹]Genetic [۱۰]Adaptive Memory Algorithm

در بخش های آینده به معرفی و توضیح بیشتر بعضی از این الگوریتم ها می پردازیم.

۱.۲ الگوریتم حریصانه برای رنگ آمیزی

این الگوریتم در واقع، یك رنگ آمیزی را برای رئوس گراف مورد نظر می سازد. به این صورت که به طور دنباله ای رئوس گراف را انتخاب کرده و کوچکترین شماره رنگ مجاز را به آن اختصاص می دهد. یعنی، به عنوان مثال، پس از انتخاب رأس v کوچکترین عدد طبیعی را که تا کنون به رئوس مجاور v اختصاص پیدا نکرده، به آن نسبت می دهد. در هر مرحله اگر امکان انتساب اعدادی که تا کنون به کار رفته اند، به رأس v نباشد، یک عدد جدید (رنگ جدید) برای اختصاص به آن انتخاب می شود.

روش های مختلفی برای ترتیب انتخاب رئوس، وجود دارند. به عنوان مثال، روش های اولین انتخاب [۱] و روش های مرتب سازی بر مبنای درجه رئوس (روش درجه مبنا) [۲] را در ادامه توضیح می دهیم:[۱۵]

---

[۱] First fit [۲] Degree based ordering

۱.۱.۲ روش اولین انتخاب(FF)

این روش آسان ترین و سریع ترین روش برای ترتیب انتخاب رئوس گراف است که در آن رئوس به دلخواه انتخاب می شوند. مرتبه زمانی این روش $O(n)$ می باشد.

۲.۱.۲ روش درجه مبنا

این روش استراتژی بهتری را برای ترتیب انتخاب رئوس، با استفاده از معیاری مشخص، ارائه می دهد که منجر به جواب های بهتری می شود. برخی از این استراتژی ها عبارتند از:

ترتیب دهی بر اساس بزرگترین درجه

روش ترتیب دهی بر اساس بزرگترین درجه[۱](LDO) در هر گام رأسی با بیشترین درجه را برای رنگ آمیزی انتخاب می کند. به طور شهودی، این روش رنگ آمیزی بهتری را بدست می دهد و زمان اجرای آن از مرتبه $O(n^2)$ می باشد.

[۱] Largest degree ordering

ترتیب دهی بر اساس درجه اشباع

درجه اشباع هر رأس، به تعداد رئوس مجاور آن که با رنگ متفاوت رنگ آمیزی شده اند، گفته می شود. به عبارت دیگر، تعداد رنگ های متفاوتی که در رئوس مجاور هر رأس به کار رفته است، درجه اشباع آن رأس نامیده می شود. در واقع، در هر گام رأسی برای رنگ آمیزی انتخاب می شود که بیشترین درجه اشباع را داشته باشد. روش ترتیب دهی بر اساس درجه اشباع[۱] (SDO) به طور شهودی، بهتر از روش بزرگترین درجه عمل می کند و مرتبه زمانی آن $O(n^3)$ می باشد.

ترتیب دهی بر اساس درجه تصادف

یك روش اصلاحی بدست آمده از روش درجه اشباع، ترتیب دهی بر اساس درجه تصادف[۲] (IDO) است. درجه تصادف هر رأس به تعداد رئوس رنگ شده مجاور آن گفته می شود. زمان اجرای این روش از مرتبه $O(n^2)$ است.

---

[۱] Saturation degree ordering [۲] Incidence degree ordering

۳.۱.۲ الگوریتم های پیشنهادی

در این قسمت دو الگوریتم بر مبنای روش های بیان شده ارائه می دهیم:

الگوریتم پیشنهادی ۱

این الگوریتم، روش ترتیب دهی بر اساس بزرگترین درجه را با روش درجه تصادف، ترکیب می کند. الگوریتم، مانند روش بزرگترین درجه ترتیب انتخاب رئوس را مشخص می کند، اما اگر در یك گام دو رأس با درجه برابر وجود داشته باشند، روش درجه تصادف برای انتخاب یکی از آن ها به کار می رود. دو معیار برای انتخاب رأسی که به دنبال رنگ کردن آن هستیم، وجود دارد:

۱. تعداد رئوس مجاور با رأس انتخابی
۲. تعداد رئوس رنگ شده مجاور با رأس انتخابی

الگوریتم پیشنهادی ۱ که ورودی و خروجی آن را می توان به صورت زیر نوشت، در شکل ۱.۲ نشان داده شده است.

- ورودی: مجموعه رئوس (گراف) به صورت $(n_1, n_2, n_3, ..., n_m)$
- خروجی: رئوس رنگ آمیزی شده و تعداد کل رنگ های استفاده شده

زمان اجرای این الگوریتم مانند روش بزرگترین درجه از مرتبه $O(n^2)$ است.

الگوریتم پیشنهادی ۲

```
while (NoOfColoredNodes < m)
{
 max=-1
 for I = 1 to m
 {
  if (!colored(n_i))
  {
   d = Degree(n_i)
   if(d>max)
   {
    max = d
    index = i
   }
   if(d = max)
    if(ID(n_i) > ID(n_index)
     //ID(Incidence Degree)
     index = i
  }
  Color(n_index)
  NoOfColoredNodes =
NoOfColoredNodes + 1
 }
}
```

شکل ۱.۲: الگوریتم پیشنهادی ۱

در این الگوریتم، دو روش درجه اشباع و بزرگترین درجه با هم ترکیب می شوند. الگوریتم، مانند روش درجه اشباع عمل می کند. اما در صورت وجود دو رأس با درجه اشباع برابر، روش بزرگترین درجه برای انتخاب یکی از آن ها به کار می رود. بنابراین، دو معیار برای انتخاب رأسی که به دنبال رنگ کردن آن هستیم وجود دارد:

۱. تعداد رنگ های به کار رفته در همسایه های رأس انتخابی
۲. تعداد رئوس مجاور با رأس انتخابی

```
while (NoOfColoredNodes < m)
{
 Max = -1
 for I = 1 to m
 {
   if (!colored(n_i))
   {
     d = SD(n_i)
     //SD( Saturated Degree)
     if(d > max)
     {
       max = d
       index = i
     }
     if (d = max)
       if(Degree(n_i) >
         Degree(n_index))
         index=i
   }
   Color(n_index)
   NoOfColoredNodes =
    NoOfColoredNodes + 1
  }
}
```

شکل ۲.۲: الگوریتم پیشنهادی ۲

الگوریتم پیشنهادی ۲ که ورودی و خروجی آن مشابه الگوریتم قبلی است، در شکل ۲.۲ نشان داده شده است. زمان اجرای این الگوریتم مانند روش درجه اشباع از مرتبه $O(n^3)$ است.

No. of Vertices	Density	FF	LDO	IDO	Proposed Algorithm 1	SDO	Proposed Algorithm 2
200	25%	20	18	18	18	17	16
200	50%	36	34	34	33	32	31
200	75%	58	55	56	55	53	52
1000	25%	64	62	63	62	58	57
1000	50%	127	123	126	122	116	115
1000	75%	217	212	214	211	204	202

شکل ۳.۲: نتایج حاصل از اجرای الگوریتم ها

۴.۱.۲ نتایج محاسباتی

در بخش های گذشته، چهار روش برای ساختن یك رنگ آمیزی برای گراف و همچنین دو الگوریتم پشنهادی که حاصل ترکیب آن ها بود، بیان کردیم. در این بخش به مقایسه الگوریتم های ذکر شده با ارائه نتایج عددی حاصل از اجرای این الگوریتم ها روی گراف های مختلف می پردازیم. شکل ۳.۲ این نتایج را در جدولی نشان می دهد. از این جدول نتایج زیر گرفته می شود:

- روش اولین انتخاب بدترین روش از لحاظ تعداد رنگ هایی است که برای رنگ آمیزی گراف استفاده می کند.
- تعداد رنگ هایی که روش بزرگترین درجه استفاده می کند، کمتر از روش درجه تصادف است.
- تعداد رنگ هایی که روش درجه اشباع استفاده می کند، کمتر از روش درجه تصادف

و بزرگترین درجه است.

- الگوریتم پیشنهادی ۱ از لحاظ تعداد رنگ هایی که استفاده می کند، بهتر از روش بزرگترین درجه است.
- الگوریتم پیشنهادی ۲ از لحاظ تعداد رنگ هایی که استفاده می کند، بهتر از روش درجه اشباع است.
- الگوریتم پیشنهادی ۲ کمترین رنگ را برای رنگ آمیزی یك گراف استفاده می کند.

۲.۲ روش های فراابتکاری

معمولاً روش های فراابتکاری ابتدا گراف داده شده را با k رنگ، رنگ آمیزی می کنند. به عبارت دیگر، این روش ها ابتدا مسأله تصمیم گیری رنگ آمیزی گراف با k رنگ را حل می کنند. سپس اگر رنگ آمیزی مطلوبی پیدا نشد، k را به دلخواه افزایش می دهند.

۱.۲.۲ روش جستجوی ممنوع

جستجوی ممنوع در حالت کلی

روش جستجوی ممنوع را می توان به این صورت بیان کرد: می خواهیم از یك جواب شدنی اولیه گام به گام به طرف جوابی حرکت کنیم که مقدار تابع هدف مسأله را به حداقل

برساند. برای این منظور، هر جواب را به صورت یك نقطه در فضای جواب نشان می دهیم و برای هر جواب مانند s یك همسایگی $N(s)$ تعریف می کنیم.

گام ابتدایی این روند، شامل شروع با یك جواب شدنی و تولید تعداد مشخص rep از جواب ها در همسایگی $N(s)$ می باشد؛ سپس از بین آن ها بهترین همسایه s^* را انتخاب کرده و به آن تغییر مکان می دهیم، خواه مقدار $f(s^*)$ بهتر از $f(s)$ باشد یا نه. تا اینجا، این روند مشابه روش جستجوی محلی است به غیر از اینکه این روش، به ما این اجازه را می دهد که به سمت یك جواب بدتر از جواب فعلی حرکت کنیم.

ویژگی جالب روش جستجوی ممنوع، ساختن یك لیست T از حرکت های ممنوع است. این ها حرکت هایی هستند که اجازه رفتن به آن ها در تکرار جاری وجود ندارد. دلیل ایجاد این لیست جلوگیری از برگشتن به جواب هایی است که در تکرارهای قبلی به آن ها رسیده ایم. هر کدام از این حرکت ها در طی تعداد مشخصی از تکرارها، در این لیست باقی می مانند. بنابراین در واقع، T یك لیست چرخشی از حرکت هاست به این ترتیب که هرگاه از s به s^* حرکت کنیم، حرکت از s^* به s در لیست ممنوع قرار می گیرد و قدیمی ترین حرکتی که در لیست قرار دارد از آن خارج می شود.

اکنون باید یك شرط توقف تعریف کنیم: در حالت کلی، می توانیم حداکثر تعداد $nbmax$ تکرار را در نظر بگیریم. در اینجا ما یك برآورد f^* از حداقل مقدار تابع هدف $f(s)$ را در نظر گرفته و به محض اینکه به اندازه کافی به f^* نزدیك شدیم (یا اگر به f^* رسیدیم) روند

را متوقف می کنیم.

به کارگیری روش جستجوی ممنوع در حل مسأله رنگ آمیزی گراف

هرتز [۱] و وِرا [۲] اولین کسانی بودند که روش جستجوی ممنوع را برای پیدا کردن رنگ آمیزی گراف های بزرگ به کار بردند. [۲]

برای گراف داده شده $G(V,E)$ یك جواب شدنی یك افراز به صورت $s=(V_1,V_2,...,V_k)$ از مجموعه رئوس V به تعداد مشخص k زیرمجموعه است. اگر $E(V_i)$ مجموعه یال هایی از گراف G باشد که هر دو سر آن ها در V_i واقع شده است، می توان تابع هدف f را تعداد یال هایی تعریف کرد که هر دو سرشان در یك V_i واقع شده اند (یعنی، همرنگ هستند).

$$f(s)=\Sigma(|E(V_i)|:i=1,\dots,k).$$

به وضوح s یك رنگ آمیزی با k رنگ برای رئوس G خواهد بود اگر و فقط اگر $f(s)=0$. در واقع می توانیم بهترین مقدار ممکن برای $f(s)$ را با $f^*=0$ برآورد کنیم که به ما یك شرط برای توقف الگوریتم می دهد.

سپس یك همسایگی s' (یك افراز kتایی دیگر از مجموعه رئوس) را به این صورت تولید می کنیم: رأس x را به طور تصادفی از بین رئوسی انتخاب می کنیم که متصل به یکی از یالهای اجتماع $E(V_1)\cup...\cup E(V_k)$ باشند. فرض کنید $x\in V_i$. حال رنگ تصادفی j را انتخاب کرده و همسایگی s' را از $s=(V_1,...,V_k)$ با تغییرات زیر می سازیم:

[۱] A. Hertz [۲] D. de Werra

$$V_j' = V_j \cup \{x\}; \quad V_i' = V_i \setminus \{x\}; \quad V_r' = V_r \; for \; r = ۱, \ldots, k; r \neq i, j.$$

پس از تولید تعداد rep همسایه از s (که منجر به حرکت های ممنوع نشوند)، بهترین همسایه را انتخاب کرده و به سمت آن حرکت می کنیم.

لیست ممنوع نیز به این صورت تولید می شود که هرگاه رأس x برای ایجاد یك جواب جدید از V_i به V_j منتقل می شود، زوج (x, i) ممنوع می شود یعنی برای چند تکرار، رأس x نمی تواند به V_i برگردد. همانطور که قبلاً بیان شد، لیست حرکت های ممنوع چرخشی است.

اکنون تکرارها را ادامه می دهیم تا زمانی که یا به جوابی مانند s که $f(s) = f^*$ یا به ماکزیمم تعداد تکرار $nbmax$ برسیم. در این حالت، در صورتی که برای آخربن جواب s داشته باشیم $f(s) > ۰$ به یك رنگ آمیزی برای گراف نخواهیم رسید.

معیار تنفس

اگر فرض کنیم $z = f(s)$ مقدار تابع هدف به ازای جواب جاری باشد، تابع $A(z)$ را به عنوان معیار تنفس برای مقدار تابع هدف گام بعد، تعریف می کنیم. این تابع به این صورت به کار گرفته می شود که اگر در یك گام حرکت به همسایه s' ممنوع باشد، اما $f(s') \leq A(z)$، آنگاه شرایط ممنوعیت را در این گام نادیده می گیریم و با آن مانند یك عضو عادی از مجموعه همسایه های تولید شده، رفتار می کنیم. در ابتدا، به ازای تمام مقادیر z قرار می دهیم: $A(z) = z - ۱$. سپس هرگاه یك جواب s' با $f(s') \leq A(f(s))$ تولید

Input $G=(V,E)$
k = number of colors
$|T|$ = size of tabu list
rep = number of neighbours in sample
nbmax = maximum number of iterations.

Initialization

Generate a random solution $s=(V_1,\ldots,V_k)$
nbiter:=0; choose an arbitrary tabu list T.

While $f(s)>0$ **and** nbiter < nbmax
generate rep neighbours s_i of s with move $s \to s_i \notin T$ or $f(s_i) \leq A(f(s))$
(as soon as we get an s_i with $f(s_i)<f(s)$ we stop the generation).

Let s' be the best neighbour generated

update tabu list T
(introduce move $s \to s'$ and remove oldest tabu move)

$s:=s'$
nbiter:=nbiter + 1

endwhile

Output If $f(s)=0$, we get a coloring of G with k colors: $V_1,\ldots,V_k$ are the color sets. Otherwise no coloring has been found with k colors.

شکل ۴.۲: الگوریتم جستجوی ممنوع

شد، قرار می دهیم $A(f(s)) = f(s') - 1$.

الگوریتم کامل روش جستجوی ممنوع برای مسأله رنگ آمیزی گراف در شکل ۴.۲ آمده است. در طول فرآیند تولید همسایه برای s اگر در یك گام همسایه s' (که در لیست ممنوع نباشد) با $f(s') < f(s)$، آنگاه به جای ادامه دادن تا تولید تعداد rep همسایه، مستقیماً از s به سمت s' حرکت می کنیم.

۲.۲.۲ الگوریتم ژنتیك

الگوریتم ژنتیک به‌عنوان یکی از روش های تصادفی بهینه یابی شناخته شده، توسط جان هالند ابداع شده [۱۸] و از اصول انتخاب طبیعی داروین برای یافتن فرمول بهینه جهت پیش‌بینی یا تطبیق الگو استفاده می‌کند.

در حالت کلی، الگوریتم ژنتیك با یك جمعیت اولیه از جواب های کاندید شروع به کار می کند. هر کدام از این جواب ها یك کروموزوم[۱] نامیده می شود و معمولاً به صورت یک رشته ساده از داده‌ها نمایش داده می‌شوند، البته انواع ساختمان داده‌های دیگر هم می‌توانند مورد استفاده قرار گیرند. به هر کروموزوم، یك ارزش [۲] نسبت داده می شود که نشان دهنده کیفیت جواب متناظر با این کروموزوم است. این انتساب ارزیابی [۳] نام دارد. هر چه یك کروموزوم ارزش بهتری داشته باشد، شانس بیشتری برای ماندن در جمعیت نسل های بعدی دارد. تعداد کروموزوم ها در هر نسل ثابت است.

همانند زندگی طبیعی، کروموزوم های فرزند (نسل جدید)از کروموزوم های والد (نسل قبلی) به دست می آیند. یك راه این است که دو والد از نسل قبل اطلاعاتی را با یکدیگر تبادل کنند و دو فرزند (دو جواب جدید) ایجاد کنند؛ این عملگر تقاطع [۴] نامیده می شود. یك راه دیگر تغییر اطلاعات مربوط به یك کروموزوم از نسل قبل و ایجاد یك کروموزوم جدید متفاوت برای نسل جدید است؛ این عملگر جهش [۵] نامیده می شود.

---

[۱] chromosome [۲] fitness value [۳] evaluation [۴] crossover [۵] mutation

```
P := ∅
For i := 1 to N do
    Generate a random solution s
    With probability p_m
       mutate s
    Evaluate s and add it to P
Sort P
For i := 1 to nb_gen do
    For j := 1 to nb_per_gen do
        Select two parents p_1 and p_2 from P
        offspring := crossover(p_1, p_2)
        With probability p_m
           mutate(offspring)
        Evaluate offspring and add it to P
    Sort P
    Cull(P, nb_per_gen)
Return the best solution from P
```

شکل ۵.۲: الگوریتم ژنتیك

بعضی کروموزوم ها بدون تغییر به نسل بعد منتقل شده و بعضی هم می میرند!

پیاده سازی الگوریتم ژنتیك روی مسأله رنگ آمیزی گراف

الگوریتم ژنتیك به صورت های مختلف می تواند پیاده سازی شود. پیاده سازی که ما در این جا انجام می دهیم، از یك جمعیت تقریباً یکنواخت استفاده می کند که در آن تعداد کمی از کروموزوم ها در هر نسل تغییر می کنند، در حالی که در بعضی از انواع الگوریتم ژنتیك در هر نسل به طور کامل، جمعیت جدیدی جایگزین جمعیت نسل قبل می شود. در شکل ۵.۲ این الگوریتم به طور خلاصه آمده است. در این الگوریتم، N اندازه جمعیت،

$nb-gen$ تعداد کل نسل ها و $nb-per-gen$ تعداد فرزندان تولید شده از طریق تقاطع والدهایی است که از بین جمعیت انتخاب می شوند. یك عملگر جهش نیز با احتمال p_m روی هر فرزند به کار گرفته می شود. در بخش های بعد چندین روش تقاطع و جهش را ارائه می دهیم. پس از هر نسل، اندازه جمعیت توسط تابع $Cull(P, nb-per-gen)$ دوباره به N برمی گردد. این کار با حذف کردن تعداد $nb-per-gen$ از بدترین جواب های جمعیت P انجام می شود.

اکنون بعضی از عناصر این الگوریتم کلی را به مسأله رنگ آمیزی گراف انطباق می دهیم. سایر عملگرها در اکثر الگوریتم های ژنتیك مشترك هستند. به عنوان مثال، مکانیزم انتخاب والدین باید به جواب های بهتر امکان انتخاب مجدد بیشتر بدهد. ما در این پیاده سازی از یك مکانیزم رتبه بندی استفاده می کنیم که در [۱۱] ارائه شده است. جواب ها بر طبق ارزششان ترتیب دهی می شوند، یعنی جواب بهتر زودتر می آید. برای انتخاب یك والد، عدد تصادفی $u \in [۰, N^{۱\backslash r}]$ تولید می شود (که پارامتر r یك عدد حقیقی در بازه $[۱, ۲]$ است). سپس جوابی که در موقعیت $\lceil u^r \rceil$ قرار دارد به عنوان والد انتخاب می شود. به راحتی دیده می شود که احتمال انتخاب والد بهتر با افزایش مقدار پارامتر انتخابی r افزایش می یابد.ما ابتدا قرار می دهیم $r = ۲$ و سپس به طور دوره ای آن را به صورت $r = ۱ + D$ تنظیم می کنیم که $D \in [۰, ۱]$ اندازه تنوع جمعیت را نشان می دهد.

کدگذاری کروموزوم ها

در این بخش دو روش برای کدگذاری کروموزوم ها و عملگرهای تقاطع و جهش مربوط به هر کدام را ارائه می دهیم.

- **کدگذاری ترتیبی** [۱]

در این روش کدگذاری، هر کروموزوم (جواب) به صورت جایگشتی از رئوس گراف نمایش داده می شود. برای ارزیابی یك جواب، الگوریتم حریصانه ای که در بخش ۱.۲ بیان شد، به کار گرفته می شود که به طور دنباله ای رئوس گراف را به ترتیبی که در جایگشت متناظر آمده اند، رنگ آمیزی می کند.

نکته قابل توجه در این الگوریتم حریصانه این است که نسبت به الگوریتم ارائه شده کمی اصلاح شده است و در نتیجه گراف مورد نظر را با تعداد رنگ ثابت k، رنگ آمیزی می کند. روند حریصانه، رئوس گراف را به ترتیب ملاقات می کند و به هر رأس اولین رنگی را که در دسترس است نسبت می دهد. در صورتی که هیچ کدام از k رنگ موجود را نتواند به رأسی اختصاص دهد، رنگی را انتخاب می کند که در بین همسایه های این رأس، کمترین تکرار را داشته باشد. بنابراین الگوریتم حریصانه، یك افراز از گراف مانند $C_1, C_2, ..., C_k$، تولید می کند که با تابع

$$f(C_1, C_2, ..., C_k) = \sum_{i=1}^{k} |E(C_i)|,$$

ارزیابی می شود که در آن $E(C_i) = \{(x, y) \in E : x, y \in C_i\}$ و $|E(S)|$ نشان

[۱] order-based encoding

دهنده اندازه مجموعه S است. در واقع هدف، یافتن افرازی است که در آن شرط $f(C_1, C_2, ..., C_k) = 0$ برقرار باشد.

الگوریتم ژنتیك، فضای جایگشت ها را جستجو و پس از به کار بردن الگوریتم حریصانه اصلاح شده روی هر کدام از جایگشت ها، بر اساس تابع fآن را ارزیابی می کند.

اکنون، عملگر های تقاطع و جهش را تعریف می کنیم که به ترتیب در شکل های ۶.۲ و ۷.۲ نشان داده شده است. در عملگر تقاطع، برای تولید جایگشت فرزند

$parent_1$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$parent_2$	2	5	8	1	9	3	7	4	6
str (step 1)	0	0	1	0	1	1	1	0	1
offspring (step 2)	–	–	**3**	–	**5**	**6**	**7**	–	**9**
offspring (step 3)	**2**	**8**	**3**	**1**	**5**	**6**	**7**	**4**	**9**

شکل ۶.۲: عملگر تقاطع یکنواخت در کدگذاری ترتیبی

before mutation	2	8	3	1	5	6	7	4	9
random positions			↑			↑			
after mutation	2	8	**6**	1	3	**5**	7	4	9

شکل ۷.۲: عملگر جهش در کدگذاری ترتیبی

$offspring$ از والدین $parent_1$ و $parent_2$، رشته str از صفر و یك ها با اندازه $|V|$ به طور تصادفی تولید می شود(گام ۱ در جدول شکل۶.۲). سپس، برای مکان iام که $str[i] = 1$، انتساب $offspring[i] = parent_1[i]$ (گام ۲ در جدول شکل۶.۲) صورت می گیرد. در نهایت، مکان های باقی مانده از فرزند توسط رئوسی که تا کنون در جایگشت قرار داده نشده اند، با ترتیبی که در $parent_2$ قرار گرفته اند، پر می شوند. این عملگر، عملگر تقاطع یکنواخت بر اساس کدگذاری ترتیبی [۱] نامیده می شود. عملگر جهش (جدول شکل ۷.۲ را ببینید) به طور تصادفی دو مکان را از جایگشت انتخاب می کند. سپس، رئوس قرار گرفته در این دو مکان و بین آن ها به طور تصادفی مجدد چیده می شوند.

- **کدگذاری رشته ای** [۲]

اکنون، یك روش کدگذاری دیگر را ارائه می کنیم که در آن یك رشته از رنگ ها به عنوان کروموزوم در نظر گرفته می شوند. کدگذاری که در قسمت قبل بیان شد (کدگذاری ترتیبی)، برای ترکیب با الگوریتم حریصانه که روی دنباله ای از رئوس عمل می کند، مناسب است. اما کدگذاری رشته ای امکان ایجاد طرح های پیوندی [۳] با روش های جستجوی فضا، مانند جستجوی محلی یا جستجوی ممنوع را فراهم می آورد.

---

[۱] Uniform order-based crossover [۲] string-based encoding [۳] hybrid schemes

فرض کنید $P = \{p_1, p_2, ..., p_N\}$ یك جمعیت از افرازهای V به k زیرمجموعه باشد. هر افراز $p \in P$ را می توان به صورت $p = \{C_1, C_2, ..., C_k\}$ نشان داد و به شکل یك رشته با اندازه $|V|$ کدگذاری کرد که در آن $s[i] = j$ اگر رأس $i \in C_j$. با استفاده از این کدگذاری سه عملگر تقاطع یك نقطه ای رشته ای [۱] دو نقطه ای رشته ای [۲] و یکنواخت رشته ای [۳] را تعریف می کنیم که به ترتیب در شکل های ۸.۲، ۹.۲و ۱۰.۲ نشان داده شده اند. عملگر تقاطع یك نقطه ای، به طور تصادفی یك نقطه

$parent_1$	1	2	3	2	1	3	2	1	1
$parent_2$	3	1	3	2	2	1	3	1	2
random position				↑					
$offspring_1$	1	2	3	2	2	1	3	1	2
$offspring_2$	3	1	3	2	1	3	2	1	1

شکل ۸.۲: عملگر تقاطع یك نقطه ای رشته ای

برشی در کروموزوم والدین انتخاب می کند و دو فرزند را به این صورت تولید می کند: فرزند اول عناصر قبل از نقطه برشی را از والد اول و عناصر بعد از آن به علاوه عنصری که در نقطه برشی قرار دارد، را از والد دوم می گیرد و فرزند دوم برعکس فرزند اول عمل می کند. عملگر تقاطع دو نقطه ای نیز بسیار مشابه عمل می کند،

[۱] String-based ۱-point crossover [۲] String-based ۲-point crossover [۳] String-based uniform crossover

$parent_1$	**1**	**2**	**3**	**2**	**1**	**3**	**2**	**1**	**1**
$parent_2$	3	1	3	2	2	1	3	1	2
random positions			↑			↑			
$offspring_1$	**1**	**2**	3	2	2	1	**2**	**1**	**1**
$offspring_2$	3	1	**3**	**2**	**1**	**3**	3	1	2

شکل ۹.۲: عملگر تقاطع دو نقطه ای رشته ای

$parent_1$	**1**	**2**	**3**	**2**	**1**	**3**	**2**	**1**	**1**
$parent_2$	3	1	3	2	2	1	3	1	2
random bit string	0	1	1	0	1	0	1	0	0
$offspring_1$	**1**	1	3	**2**	**2**	3	3	**1**	**1**
$offspring_2$	3	**2**	**3**	2	1	**1**	**2**	1	2

شکل ۱۰.۲: عملگر تقاطع یکنواخت رشته ای

با این تفاوت که برای تولید فرزندان، عناصر کروموزو م ها به جای یك نقطه، بین دو نقطه برشی جا به جا می شوند. عملگر تقاطع یکنواخت، یك رشته از صفر و یك ها با اندازه $|V|$ به طور تصادفی تولید می کند و توسط آن مشخص می کند که کدام والد رنگ هر رأس را تعیین کند. علاوه بر عملگر های تقاطع، یك عملگر جهش ساده را می توان این طور تعریف کرد که به هر رأس با یك احتمال مشخص، رنگی جدید به طور تصادفی نسبت داده شود.

علاوه بر این عملگرها، برای مسأله رنگ آمیزی گراف، یك عملگر تقاطع دیگر هم طراحی شده است که از ساختار گراف استفاده می کند و به عملگر تقاطع سازگار با گراف [۱] معروف است.

برای جواب s، مجموعه رئوس ناسازگار [۲] را به صورت زیر نشان می دهیم:

$$CN(s) = \{x \in V : \exists (x,y) \in E \quad s.t \quad s[x] = s[y]\}$$

هر رأس در صورتی که در یك والد ناسازگار و در والد دیگر سازگار باشد، در فرزند، رنگ متناظر در والدی را می گیرد که در آن سازگار باشد. اگر یك رأس در هر دو والد ناسازگار باشد، رنگی برای آن انتخاب می شود که بین رئوس مجاور با آن در هر دو والد، کمترین تکرار را داشته باشد. به رئوس باقی مانده، به طور تصادفی، رنگ یکی از والدین نسبت داده می شود.

این عملگر در الگوریتم شکل ۱۱.۲ توضیح داده شده است.

در این الگوریتم، $D \cup [0, 1]$ تابعی است که با احتمال برابر مقدار ۰ یا ۱ برمی گرداند و $argmin_{l \in S}\{a_1\}$ اندیس l^* را برمی گرداند که

$$a_{l^*} \leq a_l, \ \ \forall l \in S.$$

$I_l(S)$ نیز متغیر شمارنده مجموعه S است.

[۱] graph-adapted crossover [۲] conflicting nodes

For $x \in CN(parent_1)$ do
  If $(x \notin CN(parent_2))$
    $offspring[x] = parent_2[x]$
For $x \in CN(parent_2)$ do
  If $(x \notin CN(parent_1))$
    $offspring[x] = parent_1[x]$
For $x \in (CN(parent_1) \cap CN(parent_2))$ do
  $$offspring[x] = \underset{l \in \{1,\ldots,k\}}{\operatorname{argmin}} \left\{ \sum_{y \in N(x)} (I_l(\{parent_1[y], parent_2[y]\}) \right\}$$
For $x \in (V - (CN(parent_1) \cup CN(parent_2)))$ do
  If $(DU[0,1] = 1)$
    $offspring[x] = parent_1[x]$
  Else
    $offspring[x] = parent_2[x]$

شکل ۱۱.۲: عملگر تقاطع سازگار با گراف

۳.۲.۲ بعضی طرح های پیوندی

در حالت کلی، مشهور است که الگوریتم ژنتیك به تنهایی نتایج خیلی خوبی را برای مسائل بهینه سازی ترکیبیاتی، به دنبال ندارد. [۵] و [۷] به همین دلیل، معمول است که الگوریتم های ژنتیك را با روش های ابتکاری که خوب عمل می کنند، ترکیب می کنند. این طرح های پیوندی، اولین بار توسط ماهلنبین [۱] [۱۶] و داویس [۲] [۲۲] معرفی شد که در آن ها یك جستجوی محلی به عنوان عملگر جهش استفاده می شود. فلورنت [۳] و فرلند [۴] استفاده از جستجوی محلی و روند جستجوی ممنوع

[۱] Muhlenbien [۲] Davis [۳] C. Fleurent [۴] J.A. Ferland

مبتنی بر حافظه کوتاه مدت، را به عنوان ابزارهای تکاملی برای بالا بردن کارایی الگوریتم های ژنیك، در مسأله رنگ آمیزی گراف بررسی کردند. [۹]

۴.۲.۲ نتایج محاسباتی

در این بخش، چند نوع مختلف از الگوریتم ژنتیك را که فرم کلی آن در شکل ۵.۲ بیان شد، با استفاده از کدگذاری ها و عملگر های تقاطع و جهش مختلف، مورد مطالعه قرار می دهیم. هر الگوریتم با یك برچسب مشخص می شود:

$$GA(|P| \in \mathbf{N}, E \in \{P, S\}, CR \in \{\mathbf{۱}, \mathbf{۲}, U, A\}, MUT \in \{LS, TSn, p_m\}),$$

که پارامتر های استفاده شده را به اختصار نشان می دهد. پارامتر های برچسب به ترتیب عبارتند از: اندازه جمعیت، نوع کدگذاری(ترتیبی یا رشته ای)، عملگر تقاطع (یك نقطه ای، دو نقطه ای، یکنواخت یا عملگر سازگار با گراف) و عملگر جهش به کار رفته: جستجوی محلی، جستجوی ممنوع (n تعداد تکرار ها را نشان می دهد) و یا در صورت به کار بردن الگوریتم ژنتیك محض، نرخ جهش p_m را بیان می کنیم که در این صورت جهش متناسب با نوع کدگذاری به کار می رود. اگر جستجوی محلی یا جستجوی ممنوع به عنوان عملگر جهش استفاده شوند، روی هر کدام از اعضای جمعیت به کار می روند.

دو گروه از گراف ها برای مقایسه تأثیر فرایندها به کار می روند: گراف های تصادفی و گراف های لیتون. [۱] یك گراف تصادفی $G_{n,p}$ گرافی با n رأس را نشان می دهد که با احتمال p بین هر دو رأس، یال موجود است. گراف های لیتون، ساختار خاصی دارند که در [۱۲] توضیح داده شده است. عدد رنگی این گراف ها شناخته شده است. در اینجا چند مقایسه را به طور کلی، روی نمودار نشان می دهیم. جزئیات بیشتر به همراه نتایج عددی مربوط، در [۹] بیان شده است.

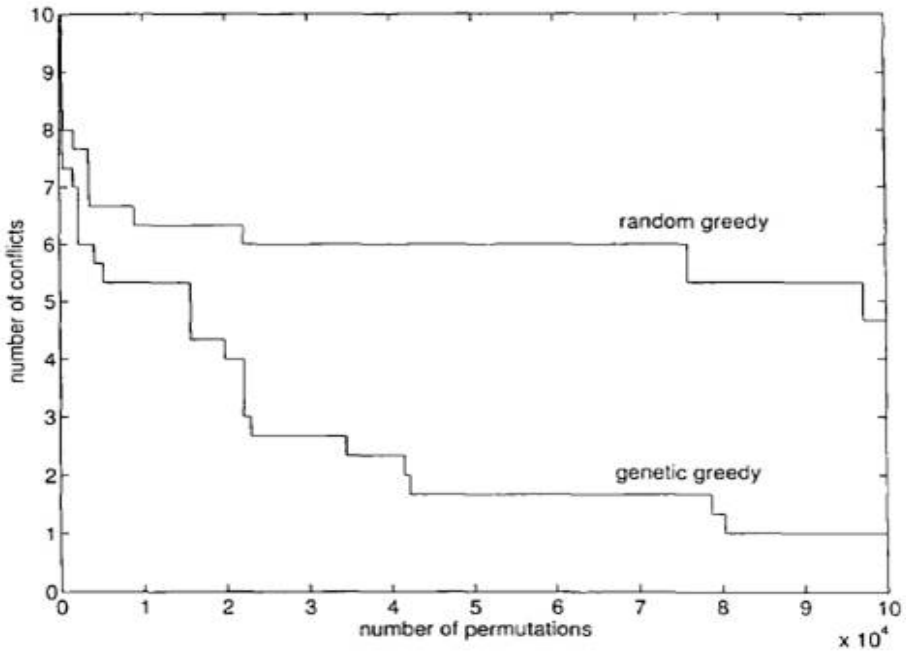
– کدگذاری ترتیبی

در نمودار شکل فوق، جستجو برای یافتن رنگ آمیزی با ۱۶ رنگ در گراف تصادفی $G_{۱۰۰,۰/۵}$ انجام شده و میانگین ۳ اجرا نشان داده شده است. برچسب الگوریتم ژنتیك به کار گرفته شده عبارت است از:

$$GA(|P| = ۱۰۰, E = P, CR = U, MUT = ۰/۰۱)$$

همچنین، در شکل ۱۲.۲ محور xها بیانگر تعداد جایگشت های تولید شده و محور yها بیانگر بهترین جواب یافت شده است. به وضوح، بهبود جواب ها توسط مولفه های ژنتیك مشاهده می شود. در شکل ۱۳.۲ جستجو بر روی یك گراف لیتون با ۴۵۰ رأس و عدد رنگی ۱۵ انجام شده است. در واقع به دنبال رنگ کردن رئوس گراف با ۱۵ رنگ هستیم. مجدداً نمودار، میانگین ۳

[۱] Leighton graphs



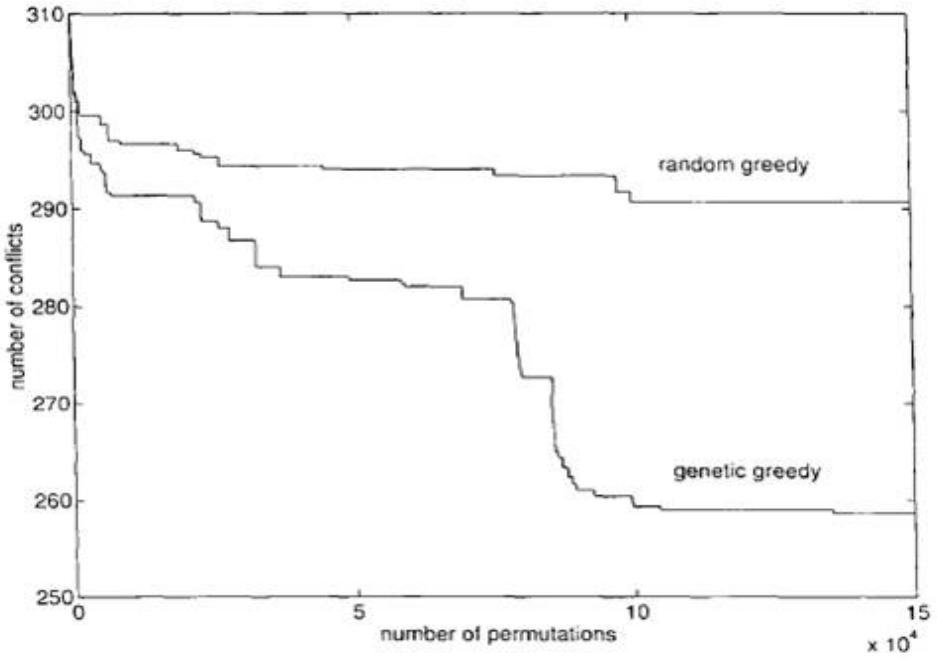
شکل ۱۲.۲: تأثیر مولفه های ژنتیك روی الگوریتم حریصانه برای گراف تصادفی $G_{۱۰۰,۰/۵}$

اجرا را نشان می دهد و برچسب الگوریتم ژنتیك عبارت است از:

$$GA(|P| = ۵۰, E = P, CR = U, MUT = ۰/۰۱)$$

نکته قابل توجه این است که با وجود اینکه در دو مثال فوق، الگوریتم ژنتیك بهبود فراوانی در جواب ها ایجاد می کند، اما مدت زمان زیادی صرف اجرای الگوریتم می شود. به طور مثال، در شکل ۱۳.۲ هر اجرا بیش ۱۲ ساعت زمان می برد و در نهایت نیز جوابی تولید می کند که خیلی از جواب بهینه فاصله دارد.

- **کدگذاری رشته ای**



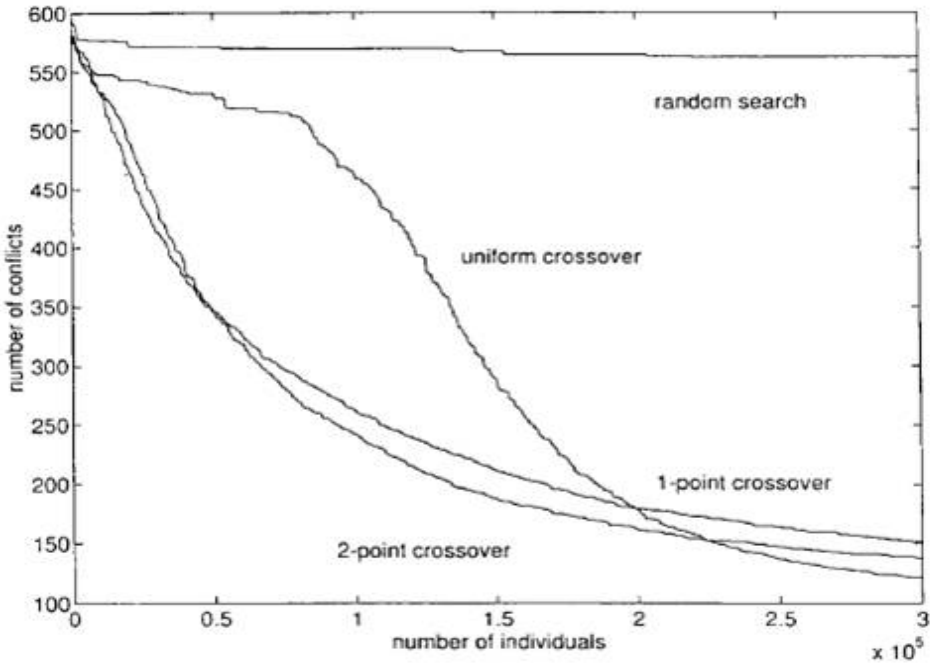
شکل ۱۳.۲: تأثیر مولفه های ژنتیك روی الگوریتم حریصانه برای گراف لیتون

مجدداً از گراف تصادفی $G_{۳۰۰,۰/۵}$ (شکل ۱۴.۲) و گراف لیتون (شکل ۱۵.۲) برای توضیح رفتار الگوریتم ژنتیك بر اساس کدگذاری رشته ای با عملگر های تقاطع مختلف، استفاده می کنیم و مجدداً بهبود آشکاری را در جستجوی تصادفی که به آسانی جواب ها را تولید و ارزیابی می کند، مشاهده می کنیم.

در گراف تصادفی $G_{۳۰۰,۰/۵}$ با برچسب

$$GA(|P| = ۱۰۰۰, E = S, CR = \{۱, ۲, U\}, MUT = ۰/۰۱)$$

به دنبال یافتن رنگ آمیزی با ۳۴ رنگ هستیم. مدت زمان اجرا برای تولید ۳۰۰۰۰۰ نسل در شکل ۱۴.۲ بیش از ۸ ساعت است. برای گراف لیتون



شکل ۱۴.۲: تأثیر عملگرهای تقاطع مختلف در گراف $G_{۳۰۰,۰/۵}$

(شکل ۱۵.۲) با ۴۵۰ رأس و عدد رنگی ۱۵ از الگوریتم ژنتیك با برچسب

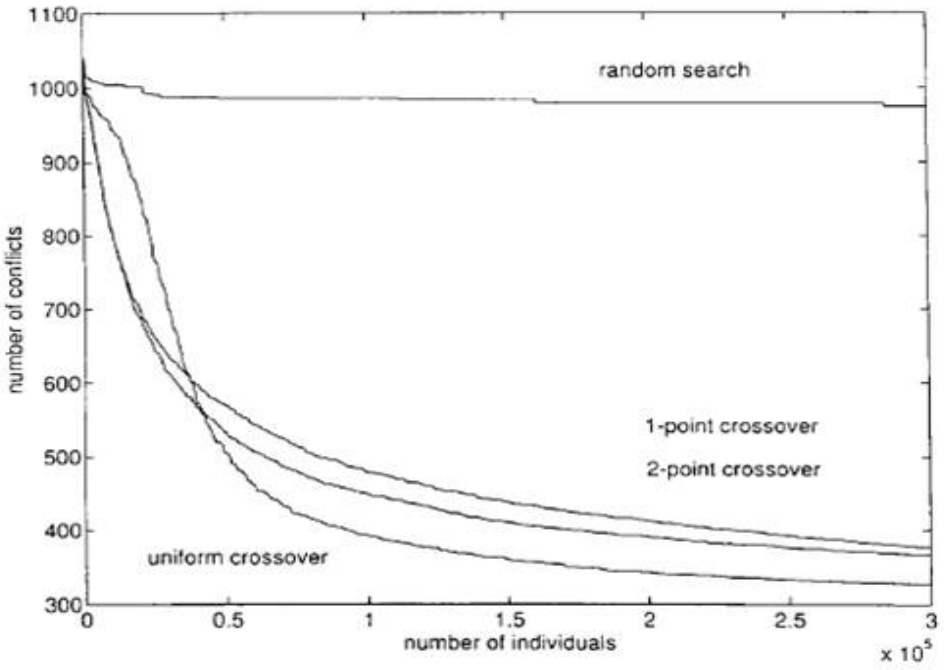
$$GA(|P| = ۴۰۰, E = S, CR = \{۱, ۲, U\}, MUT = ۰/۰۱)$$

استفاده کرده ایم و زمان اجرا نیز مشابه شکل ۱۴.۲ می باشد.

این نتایج شاید در مقایسه با نتایج حاصل از سایر روش های ابتکاری، خیلی موثر نباشد، اما به هر حال، به وضوح نمایانگر این است که عملگرهای ژنتیك می توانند در جستجوی رنگ آمیزی های خوب، کمك کننده باشند.

- **کدگذاری رشته ای ترکیب شده با جستجوی محلی**

اصلی ترین ایراد روند جستجوی محلی این است که در صورت رسیدن به یك

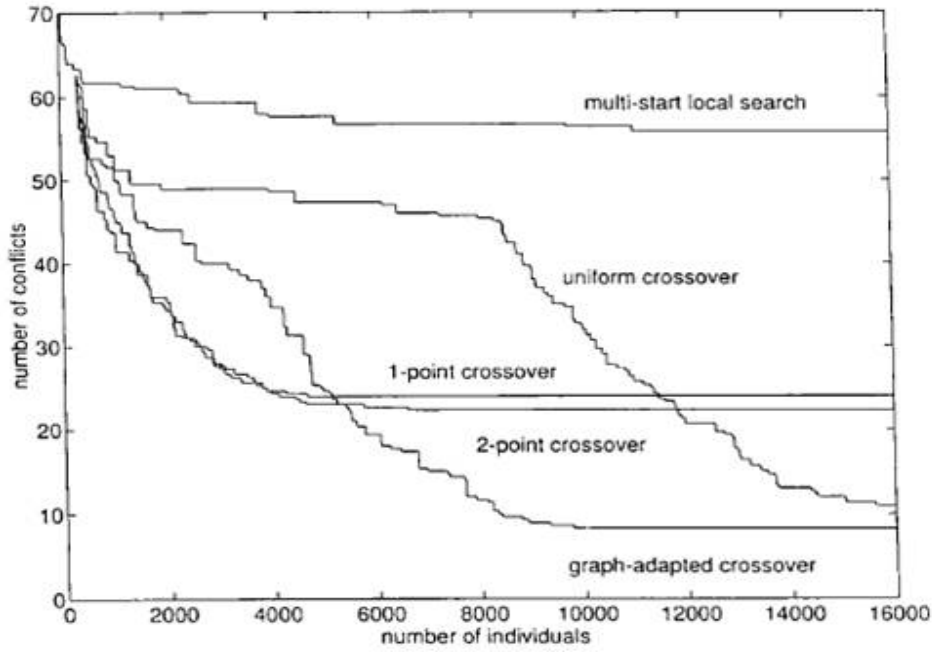


شکل ۱۵.۲: تأثیر عملگرهای تقاطع مختلف در گراف لیتون

بهینه محلی، یعنی در صورتی که هیچ همسایه بهتری از جواب جاری موجود نباشد، متوقف می شود. از نظر تاریخی، اولین رویکرد برای حل این مشکل، شروع مجدد روند جستجوی محلی از چندین جواب تصادفی بود. در این قسمت، ما تأثیر ترکیب الگوریتم ژنتیك با کدگذاری رشته ای را با الگوریتم جستجوی محلی مشاهده می کنیم. برای این منظور، عملگر جهش در شکل ۵.۲ را با یك روند جستجوی محلی روی هر کدام از اعضای جمعیت، متناظر می کنیم.

شکل های ۱۶.۲ و ۱۷.۲ نتایج حاصل از این ترکیب را با الگوریتم جستجوی

محلی با چند نقطه شروع، مقایسه می کنند. در شکل ۱۶.۲، به دنبال یافتن

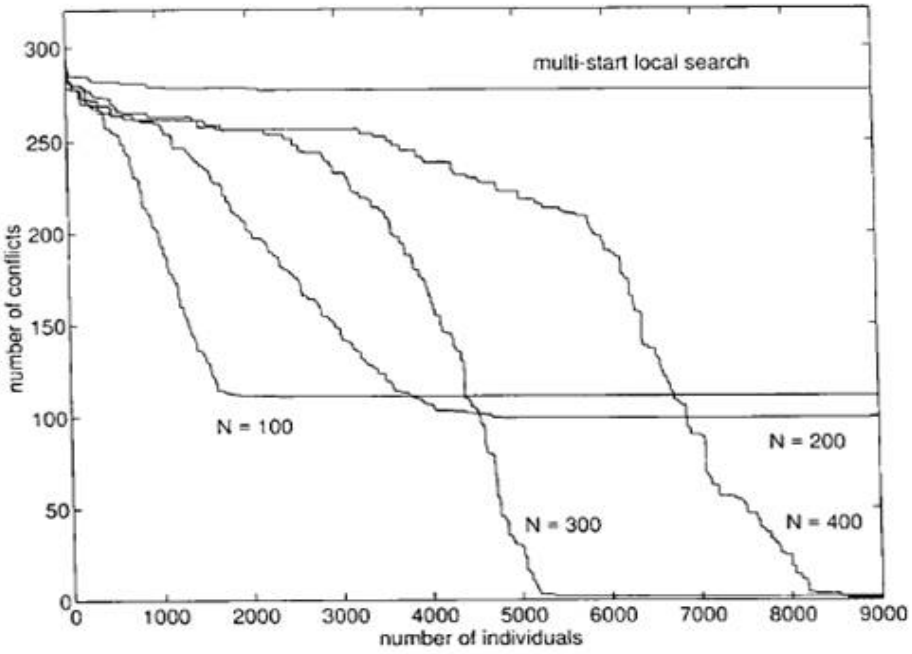


شکل ۱۶.۲: تأثیر عملگرهای تقاطع مختلف روی جستجوی محلی

یك رنگ آمیزی با ۳۴ رنگ برای گراف تصادفی $G_{۳۰۰,۰/۵}$ با برچسب

$$GA(|P| = ۳۰۰, E = S, CR = \{۱, ۲, U, A\}, MUT = LS)$$

هستیم. ملاحظه می شود که الگوریتم ترکیبی به ازای تمام عملگرهای تقاطع، به وضوح بهتر از جستجوی محلی با چند نقطه شروع، عمل می کند. توجه کنید که عملگر تقاطع یکنواخت همچنان بر دو عملگر تقاطع یك نقطه ای و دو نقطه ای، غلبه دارد. عملگر تقاطع سازگار با گراف نیز نتایج بسیار خوبی را به دنبال دارد. شکل ۱۶.۲ میانگین سه اجرا را نشان می دهد و هر اجرا حدود



شکل ۱۷.۲: تأثیر جمعیت های مختلف روی الگوریتم ترکیبی ژنتیك و جستجوی محلی

۱۴ ساعت به طول می انجامد.

همانطور که قبلاً ذکر شد، افزایش اندازه جمعیت، در حالت کلی، همگرایی الگوریتم ژنتیك را کندتر می کند اما جواب های بهتری تولید می شوند. در شکل ۱۷.۲ ، تلاش می کنیم تا همان گراف لیتون به کار رفته در شکل های ۱۳.۲ و ۱۵.۲ را توسط الگوریتم ترکیبی با برچسب

$$GA(|P| = \{۱۰۰, ۲۰۰, ۳۰۰, ۴۰۰\}, E = S, CR = U, MUT = LS)$$

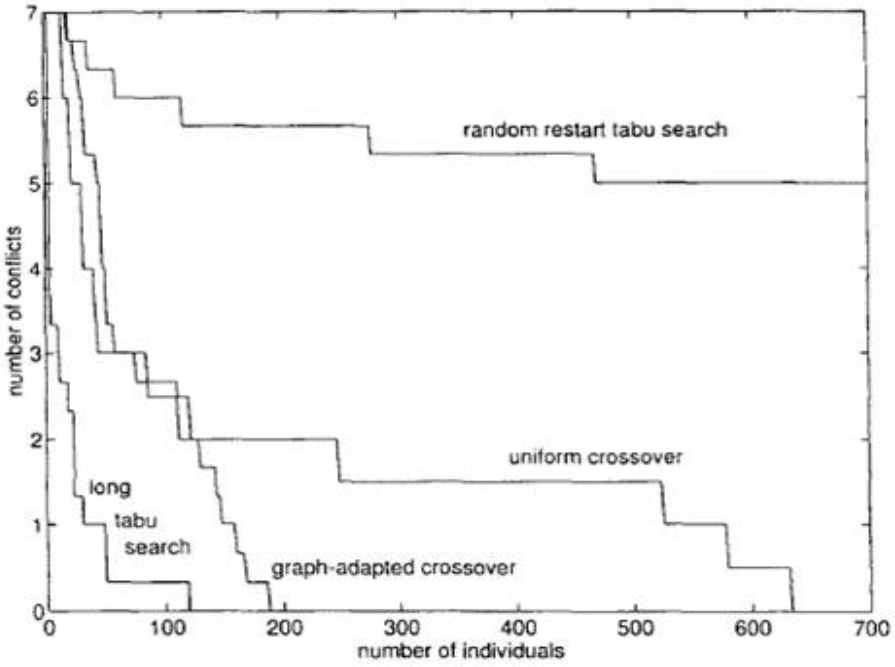
به طور بهینه رنگ آمیزی کنیم. نمودار این شکل، رفتار جمعیت ها با اندازه های مختلف، با الگوریتم ترکیبی ژنتیك-جستجوی محلی با استفاده از عملگر

تقاطع یکنواخت، را نشان می دهد. توجه کنید که جمعیت با اندازه ۳۰۰ مناسب برای تولید جواب بهینه در حدود ۶ ساعت است. جمعیت با اندازه ۴۰۰ نیز به جواب بهینه منجر می شود، اما همگرایی آن کندتر است.

- **کدگذاری رشته ای ترکیب شده با جستجوی ممنوع**

جستجوی ممنوع که در بخش ۱.۲.۲ به طور مفصل توضیح داده شد، یك روش قدرت مند است که جواب های بسیار خوبی برای مسأله رنگ آمیزی گراف می دهد. به عنوان مثال، با یك روش جستجوی ممنوع اصلاح شده [۹] می توان گراف تصادفی $G_{۱۰۰,۰/۵}$ (شکل ۱۲.۲) را در کمتر از یك ثانیه با ۱۶ رنگ، و در ۱۰ ثانیه با ۱۵ رنگ، رنگ آمیزی نمود. برای گراف های بزرگتر و مشکل تر، تنوع و قوت جواب ها اهمیت پیدا می کند. در این قسمت، نشان می دهیم که عملگرهای ژنتیك می توانند به عنوان یك روش تأمین حافظه بلند مدت، برای روند جستجوی ممنوع به کار روند. شکل ۱۸.۲ ،میانگین رفتار تعدادی از روش ها را برای رنگ آمیزی گراف تصادفی $G_{۳۰۰,۰/۵}$ با ۳۴ رنگ، نشان می دهد: یك روش جستجوی ممنوع با تعداد تکرار بسیار زیاد (۲۰۰۰۰۰۰ تکرار)، روش جستجوی ممنوع با چندین نقطه شروع تصادفی (۷۰۰ جستجو با تعداد تکرار ۲۰۰۰ در هر کدام)، و الگوریتم ترکیبی ژنتیك-جستجوی ممنوع با برچسب

$$GA(|P| = ۱۰, E = S, CR = \{U, A\}, MUT = TS۲۰۰۰).$$



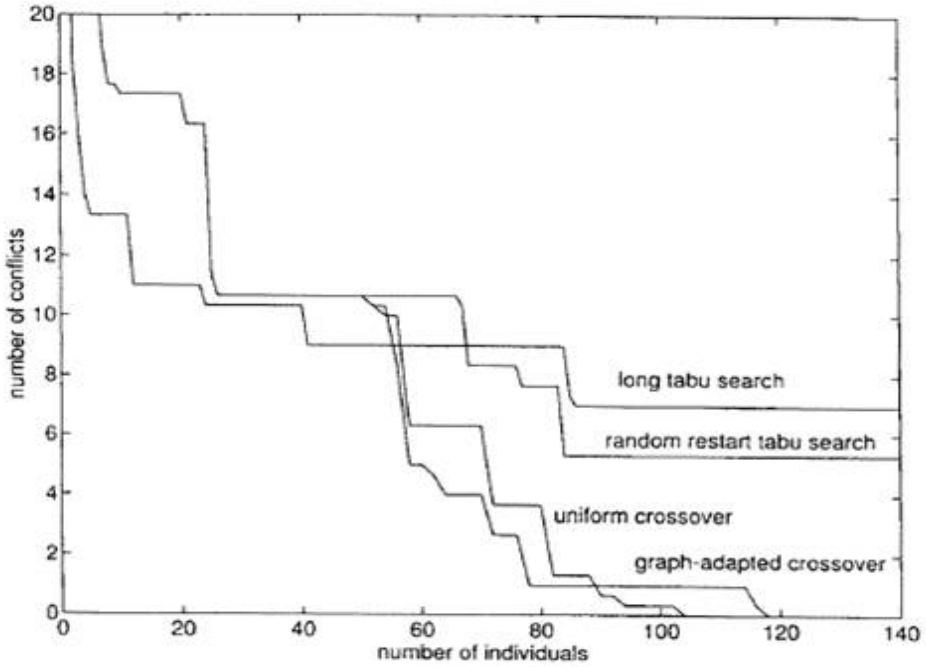
شکل ۱۸.۲: تأثیر الگوریتم ژنتیك روی جستجوی ممنوع در گراف تصادفی

مشاهده می شود که جستجوی ممنوع با تعداد تکرار بسیار زیاد، موثرترین روش است که به طور متوسط در ۳۴۳ ثانیه به رنگ آمیزی با ۳۴ رنگ، می رسد. الگوریتم ترکیبی نیز به جواب بهینه می رسد اما مدت زمان بیشتری را صرف می کند.

برای گراف لیتون به کار رفته در قسمت های قبلی نیز جستجو های طولانی تر روی هر کدام از اعضای جمعیت به کار می روند. در شکل ۱۹.۲ از الگوریتم ژنتیك با برچسب

$$GA(|P| = ۵۰, E = S, CR = \{U, A\}, MUT = TS۱۰۰۰۰)$$

استفاده کرده ایم. برای هر دو عملگر تقاطع یکنواخت و سازگار با گراف،



شکل ۱۹.۲: تأثیر الگوریتم ژنتیك روی جستجوی ممنوع در گراف لیتون

تعداد کمی نسل برای تولید جواب بهینه لازم است(در کمتر از ۲ ساعت). علاوه بر این، در این حالت الگوریتم ژنتیك ترکیبی، کارایی روش جستجوی ممنوع ابتدایی را افزایش می دهد. در واقع، با شروع از ۵ جواب تصادفی، روش جستجوی ممنوع اصلاح شده، قادر به رسیدن به جواب بهینه حتی پس از ۲۰۰۰۰۰۰۰ تکرار برای هر نقطه شروع، نیست.

مراجع

[1] A. Cayley, On the colorings of maps. Proc. Royal Geographical Society 1, 259–261, 1879.

[2] A. Hertz, D. de Werra, Using Tabu Search Techniques for Graph Coloring. Computing, 39(4), 345-351, 1987.

[3] A. Hertz, M. Plumettaz, N. Zufferey, Variable space search for graph coloring, Discrete Applied Mathematics, 156(13), 2551-2560, 2008.

[4] A. Mehrotra and M. Trick, A column generation approach for graph coloring, INFORMS J. on Computing, Vol. 8, No. 4, pp. 344-353, 1996.

[5] B. Carter, K. Park, How good are genetic algorithms at finding large

cliques: An experimental study, Technical Report BU-CS-93-015, Boston University, 1994.

[6] B. Korte and J. Vygen: Combinatorial optimization theory and algorithms, Springer,fourth edition, 2008.

[7] B.R. Fox, M.B. McMahon, Genetic operators for sequencing problems, Foundations of Genetic Algorithms, ed. G.J.E. Rawlings, 284-300, 1992.

[8] C. Avanthay, A. Hertz, N. Zufferey, A variable neighborhood search for graph coloring. European Journal of Operational Research, 151, 379-388, 2003.

[9] C. Fleurent and J.A. Ferland, Genetic and hybrid algorithms for graph coloring. Annals of Operations Research. 63, 437-463, 1995.

[10] D. Brelaz, New methods to color the vertices of a graph, Communications of the ACM 22,pp 251-256, 1979.

[11] D.E. Tate, A.E. Smith, A genetic approach to the quadratic assignment problem, Computers and Operations Research, 22, 73-83, 1995.

[12] F. Leighton, A graph coloring algorithm for large scheduling problems, J. Res. Nat. Bureau of Standards, 84, 489- 505, 1979.

[13] Garey, M., and Johnson, D. Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.

[14] G. Chwrtrand, Applied and Algorithmic Graph Theory, New York, Mcgraw Hill college,1993.

[15] H. Al-Omari, K. Sabri, New Graph Coloring Algorithms, American Jornal of Mathematics and Statistics 2(4), 739-741, 2006.

[16] H. Muhlenbein, M. Gorges-Schleuter and O. Kramer, Evolution algorithms in combinatorial optimization, Parallel Comp. 7, 65-88, 1988.

[17] I. M´endez D´ıaz and P. Zabala, A cutting plane algorithm for graph

coloring, Discerete Applied Mathematics, 156(2), 159-179, 2008.

[18] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan press, Ann Arbor, 1975.

[19] J. P. Hamiez, J. K. Hao, Scatter Search for Graph Coloring, P. Collet, C. Fonlupt, J. K. Hao, E. Lutton, M. Schoenauer, Artificial Evolution, 2310, 168-179, 2001.

[20] K. Appel and W. Haken, Every planar map is four colorable. Part I. Dischargin g, Illinois J. Math. 21, 429–490, 1977.

[21] K. Appel, W. Haken and J. Koch, Every planar map is four colorable. Part II. Reducibility, Illinois J. Math. 21, 491–567, 1977.

[22] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[23] M. Caramia, P. Dell'Olmo, Coloring graphs by iterated local search traversing feasible and infeasible solutions. Discrete Applied Mathe-

matics, 156(2), 201-217, 2008.

[24] M. Caramia, P. Dell'Olmo, G. F. Italiano, Improved local search for graph coloring, Journal of Discrete Algorithms, 4(2), 277-298, 2006.

[25] M. Chams, A. Hertz, and D. De Werra . Some experiments with simulated annealing for coloring graphs, European Journal of Operational Research, 32: 260-266, 1987.

[26] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas, A new proof of the four color theorem, Electron. Res. Announc. Amer. Math. Soc. 2, 17–25, 1996.

[27] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas, The four color theorem, J. Combin. Theory Ser. B. 70, 2–44, 1997.

[28] P. Galinier, A. Hertz, N. Zufferey, An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics, 156(2), 267-279, 2008.

[29] P. S. Segundo, A new DSATUR-based algorithm for exact vertex coloring, Computers and Operations Research, 39(7), 1724-1733, 2011.

[30] P. Zabala, I. M´endez-D´ıaz, A branch-and-cut algorithm for graph coloring, Discerete Applied Mathematics, 154(5), 826–847, 2006.

[31] R. Karp, Reducibility among combinatorial problems, Complexity of computer computations, eds. R.Miller and J. Thatcher, 1972, 85-103

[32] T. N. Bui, T. H. Nguyen, C. M. Patel, K. A. T. Phan, An ant-based algorithm for coloring graphs. Discrete Applied Mathematics, 156(2), 190-200, 2008.