هفتمین کنفرانس مهندسی برق ایران
مرکز تحقیقات مخابرات ایران
۲۷-۲۹ اردیبهشت ۱۳۷۸

الگوریتم های تصادفی از نوع لاس وگاس برای مسئله تناظرگراف*

حمید بیگی **محمد رضا میبدی**

دانشکده مهندسی کامپیوتر
دانشگاه صنعتی امیر کبیر
تهران، ایران

چکیده

گراف یک ساختار بسیار مهم است که دارای کاربردهای فراوا نی میباشد. یکی از مسایل مهم در گراف، مسئله تناظرگراف میباشد. مسئله تناظر گراف دارای کاربردهای متعددی ازجمله صحت مدارهای VLSI[۱][۲]، صحت مدارهای PC- Board [۳]، معادل بودن دو برنامه [۴]، شناسایی الگو[۵] و بینایی ماشین [۶][۷] میباشد. تاکنون الگوریتمی با پیچیدگی زمانی چند جمله ای برای حل این مسئله گزارش نشده است. برای گراف با n گره، الگوریتم Backtracking در بدترین حالت دارای زمان اجرایی از مرتبه نمایی است که برای گراف های بزرگ مقرون بصرفه نمیباشد. بهمین جهت برای افزایش کارایی الگوریتم تناظر گراف ، از الگوریتم های تقریبی از جمله از الگوریتم بزرگترین Clique، روشهای Relaxation، جستجوی درخت، آنالیز جبری، شبکه های عصبی و الگوریتم های ژنتیکی استفاده شده است که منجربه کاهش قابل ملاحظه ای در زمان مورد نیاز برای حل مسئله تناظرگراف گردیده است. بالا بودن زمان مورد نیاز برای حل مسئله تناظر گراف بعلت نیاز به جستجوی تمام فضای جواب ها میباشد. الگوریتم های تصادفی یک روش برای کم کردن زمان جستجوا ست. این الگوریتم ها دارای دو مزیت عمده کارایی و سادگی میباشند. این بدین معنی است که الگوریتم های تصادفی برای اغلب مسایل، بسیار سریعتر از الگوریتم های قطعی شناخته شده اجرا میشوند و همچنین توصیف و پیاده سازی الگوریتم های تصادفی بمراتب ساده تر از تو صیف و پیاده سازی الگوریتم های قطعی با کارایی مشابه است. در این مقاله شش الگوریتم تصادفی از نوع لاس و گاس با استفاده از جستجوی تصادفی برای مسئله تناظر گراف ارائه شده است. این الگوریتم ها هم برای گراف های بدون اغتشاش و هم برای گراف های دارای اغتشاش قابل استفاده اند. تا آنجایی که نگارندگان این مقاله اگاهی دارند تا کنون الگوریتم تصادفی برای مسئله تناظر گراف گزارش نشده است. آزمایشهای مختلف نشان میدهند که برای گراف با اندازه n، بعضی از الگوریتم های پیشنهادی دارای پیچیدگی از مرتبه $O(n^4)$ و بعضی دیگر دارای پیچیدگی از مرتبه $O(n^3)$ میباشند که درمقایسه با الگوریتم Backtracking بهبود قابل ملاحظه ای را در مرتبه بزرگی نشان میدهند.

کلمات کلیدی: تناظرگراف، تناظر گراف با تصحیح خطا، الگوریتم های تصادفی، مسائل ذاتا مشکل

۱- مقدمه

یک گراف بصورت زوج $G = (V, E)$ نمایش داده میشود که V مجموعه گره های گراف و E مجموعه کمانهای آن میباشد. بسیاری از الگوریتم های گراف دارای پیچیدگی از مرتبه نمایی میباشند و با افزایش اندازه گراف کارایی خود را از دست میدهند. یکی از مسایل مهم در گراف، مسئله تناظرگراف[۱] میباشد که بصورت زیر تعریف میگردد. طبق تعریف ، دو گراف $G_1 = (V_1, E_1)$ و $G_2 = (V_2, E_2)$ را متناظر میگویند اگر یک تابع یک به یک و پوشا $f: V_1 \rightarrow V_2$ وجود داشته باشد که برای هر کمان (u, w) موجود در گراف G_1 ، کمان $(f(u), f(w))$ در گراف G_2 وجود داشته باشد. این مسئله از گروه مسایل NP میباشد ولی هنوز تعلق آن به گروه مسایل NP-Complete اثبات نشده است. بدین معنی که تاکنون الگوریتمی با پیچیدگی زمانی چند جمله ای برای حل این مسئله گزارش نشده است. شکل ۱ الگوریتم Backtracking برای تشخیص تناظرگراف را نشان میدهد. برای گراف با اندازه n این الگوریتم در بدترین حالت دارای زمان اجرایی از مرتبه $O(n!)$ میباشد.

مسئله تناظر گراف دارای کاربردهای بسیاری از جمله صحت مدارهای VLSI[۱][۲]، صحت مدارهای PC-Board [۳]، معادل بودن دو برنامه[۴]، شناسایی الگو[۵] و بینایی ماشین [۶][۷] میباشد. مثلا در کاربردهای بینایی ماشین ، شی و الگو توسط دو گراف مختلف نشان داده میشوند. پس از تبدیل اشیا به گراف های متناظر، مسئله تشخیص اشیا به مسئله تناظرگراف تبدیل میشود. اما بدلیل وجود اغتشاش ، دو گراف بطور دقیق با

---

* قسمتی از این پروژه با حمایت مالی مرکز تحقیقات فیزیک نظری- پژوهشکده سیستم های هوشمند انجام گردیده است.

1- Graph Isomorphism

هم متناظر نمیباشند. در اینگونه موارد هدف پیدا نمودن تناظری است که کمترین خطا را تولید نماید. به این تناظر، تناظر بهینه میگویند و به مسئله پیدا کردن تناظر بهینه دو گراف غیر متناظر، تناظر گراف با تصحیح خطا[1] گفته میشود[۶][۸]. در این حالت مسئله بسیار پیچیده تر ازحالت قبل میگردد و درگروه مسائل NP-Hard قرار دارد. برای افزایش کارایی الگوریتم تناظر گراف، الگوریتمهای تقریبی ازجمله الگوریتم بزرگترین Clique[۸]، روشهای Relaxation [۷]، جستجوی درخت[۹]، آنالیز جبری[۸]، شبکه های عصبی [۱۰] و یا الگوریتم های ژنتیکی[۸][۱۱] ارائه شده اند که منجر به افزایش قابل ملاحظه ای در کارایی الگوریتم تناظر گراف گردیده است.

```
function GraphIsomorphism (V_G, V_H)
   Let k = |V_H|  ,  n = |V_G|
   while k > 0 do
      f = {(u_n, v_k)} ∪ GraphIsomorphism (V_G - {u_n}, V_H - {v_k})
      if Error of mapping f is zero then
         return f
      end if
      dec (k)
   end while
   return φ
end GraphIsomorphism
```

شکل ۱: الگوریتم تشخیص تناظر گراف

برای پیدا کردن جواب مسایل ذاتا مشکل ، همچون تناظر گراف، الگوریتم نیاز به جستجوی فضای مسئله دارد. در الگوریتم تناظر گراف ، زمان اجرا بطور مستقیم به تعداد نگاشت های فضای مسئله وابسته میباشد. اگر چه درصد بسیار زیادی از این نگاشت ها ، جواب مسئله نمیباشد اما الگوریتم جستجو تک تک آنها را بررسی خواهد کرد. اگر به طریقی بتوان با آزمایشهای کمتری تعداد نگاشت های بیشتری ازاین فضا را بررسی نمود میتوان مرتبه بزرگی الگوریتم را کاهش داد. یکی از راههای پایین آوردن تعداد آزمایشها، استفاده از الگوریتم های تصادفی [۱۲](الگوریتم هایی که در حین اجرا از انتخاب های تصادفی استفاده میکنند) میباشد. الگوریتم های تصادفی دارای دو مزیت عمده کارایی و سادگی میباشند. کارایی الگوریتم های تصادفی باین معناست که برای بسیاری از مسایل، الگوریتم های تصادفی بسیار سریعتر از الگوریتم های قطعی شناخته شده اجرا میشوند. مفهوم سادگی الگوریتم های تصادفی باین معناست که توصیف و پیاده سازی الگوریتم های تصادفی بسیار ساده تر از تو صیف و پیاده سازی الگوریتم های قطعی با کارایی مشابه است.

در این مقاله شش الگوریتم تصادفی از نوع لاس وگاس که از روش جستجوی تصادفی استفاده میکنند برای مسئله تناظر گراف ارائه شده است. تا آنجایی که نگارندگان این مقاله اگاهی دارند تا کنون الگوریتم تصادفی برای مسئله تناظر دو گراف گزارش نشده است. آزمایشهای مختلف نشان میدهند که برای گراف با اندازه n، بعضی از این الگوریتم ها دارای پیچیدگی از مرتبه $O(n^4)$ و بعضی دیگر دارای پیچیدگی از مرتبه $O(n^3)$ میباشند که در مقایسه با الگوریتم Backtracking بهبود قابل ملاحظه ای را در مرتبه بزرگی نشان میدهد.

بخش های بعدی مقاله بصورت زیر سازماندهی شده ا ست. مقدمه ای بر الگوریتم های تصادفی در بخش ۲ آمده است. در بخش ۳ الگوریتم های تصادفی پیشنهادی برای تناظر دو گراف ارائه شده است. در بخش ۴ نتایج شبیه سازیها و در پایان نتیجه گیری آمده است.

۲- الگوریتم های تصادفی

الگوریتم های تصادفی به الگوریتم هایی گفته میشود که در حین اجرا از انتخاب های تصادفی استفاده میکنند. این الگوریتم ها دارای دو مزیت عمده کارایی و سادگی میباشند. کارایی الگوریتم های تصادفی باین معناست که برای بسیاری از مسایل، الگوریتم های تصادفی بسیار سریعتر از الگوریتم های قطعی شناخته شده اجرا میشوند. مفهوم سادگی الگوریتم های تصادفی باین معناست که توصیف و پیاده سازی الگوریتم های تصادفی بسیار ساده تر از تو صیف و پیاده سازی الگوریتم های قطعی با کارایی مشابه است.

الگوریتم های تصادفی را میتوان به چها دسته عمده الگوریتم های عددی ، الگوریتم های مونتکارلو، الگوریتم های شروود و الگوریتم های لاس وگاس تقسیم نمود که در ادامه بطور مختصر به شرح هر دسته از این الگوریتم ها میپردازیم[۱۳].

الگوریتم های عددی: الگوریتم های تصادفی برای اولین بار در مسائل عددی استفاده شدند که الگوریتم قطعی برای حل آنها وجود نداشت. برای مثال، برای تخمین طول یک صف در یک سیستم، حل بسته یا حلی که توسط الگوریتم های قطعی ممکن باشد وجود ندارد و معمولا از الگوریتمهای تصادفی برای این منظور استفاده میشود. جواب های بدست آمده توسط این دسته ازالگوریتم های تصادفی تقریبی هستند اما دقت متوسط آنها با افزایش زمان اجرا، افزایش میابد.

الگوریتم های مونتکارلو: این نوع الگوریتم ها در مورد مسائل تصمیم گیری بکار برده میشوند و همیشه به جواب همگرا میشوند. اما جواب های تولید شده توسط این الگوریتم ها الزاما صحیح نیستند. در این الگوریتم ها، احتمال صحت جواب تولید شده را میتوان با تکرار اجرای الگوریتم افزایش داد. تصمیم گیری با کارایی بالا در مورد تشخیص صحت جواب تولید شده، اساسی ترین مشکل این دسته از الگوریتم های تصادفی میباشد.

1- Error Correcting Graph Isomorphism
2- Approximate Algorithms
3- Interactable Problems
4- Order of Magnitude
5- Randomized Algorithms
6- Monte Carlo Algorithms
7- Sherwood Algorithms
8- Las Vegas Algorithms

الگوریتم های شروود: این دسته همیشه به جواب صحیح همگرا میشوند و در مواردی استفاده میشوند که برای مسئله مورد نظر، الگوریتم قطعی وجود داشته باشد که دارای متوسط زمان اجرای خوب اما در بدترین حالت دارای زمان اجرای بدی باشد. با اضافه نمودن انتخاب های تصادفی در این نوع الگوریتم ها، الگوریتم های شروود تولید میشوند که قادر به کاهش تفاوت زمان اجرای متوسط و زمان اجرای بدترین حالت هستند. هر چند زمان اجرای این الگوریتم ها دارای توزیع یکنواخت است اما دارای متوسط زمان اجرای کمتری نسبت به الگوریتم های قطعی مشابه نیستند ولی کران بالای زمان اجرای این الگوریتم ها مشخص و قابل پیش بینی است.

الگوریتم های لاس وگاس: این نوع از الگوریتم های تصادفی هیچوقت جواب غلط تولید نمیکنند اما ممکن است همیشه به جواب همگرا نشوند. این دسته از الگوریتم های تصادفی از الگوریتم های قطعی سریعترند و در مواردی که الگوریتم قطعی مناسب برای حل مسئله وجود ندارد بسیار مناسبند. کران بالای زمان اجرای این الگوریتم ها مشخص نیست زیرا ممکن است رشته متوالی از اعداد تصادفی انتخاب شوند که رسیدن به جواب را غیر ممکن سازند. احتمال همگرایی به جواب در این الگوریتم ها را با اجرای متوالی آنها میتوان افزایش داد. در صورتی یک الگوریتم لاس وگاس را صحیح میگویند که احتمال تولید جواب صحیح برای هر ورودی بزرگتر از صفر باشد.

روشهای متعددی برای طراحی الگوریتم های تصادفی ارایه شده است که میتوان آنها را به پنج دسته عمده زیر تقسیم نمود[۱۲] برای حل یک مسئله میتوان از ترکیبی از این روش ها استفاده نمود در نتیجه این روشها الزاما روشهای مستقلی نیستند.

تصادفی نمودن ورودیها[1]**:** نظم موجود در ورودیها، زمان اجرای الگوریتم ها را از زمان اجرای متوسط دور میکند. تصادفی نمودن ورودیها عبارتست از تغییر ترتیب ورودیها برای برهم زدن هر گونه نظم در ورودیها. این عمل برای نزدیک نمودن زمان مورد انتظار اجرای الگوریتم به زمان اجرای متوسط میباشد. این روش در مسایلی موثر واقع میشود که دارای زمان اجرای متوسط خوب هستند ولی بدلیل وجود الگوهای نامناسب در ورودی دارای زمان اجرای نامناسب در بدترین حالت میباشند. الگوریتم تصادفی Quick Sort یک نمونه از الگوریتمی میباشد که با این روش طراحی شده است.

جستجوی تصادفی: جستجوی تصادفی، یکی از متداولترین روش های طراحی الگوریتم های تصادفی میباشد. حل بسیاری از مسایل ، برای پیدا نمودن مقدار پارامتر مورد نظر، از طریق جستجوی فضای پارامترها انجام میشود. اگر خصوصیت پارامترها بسادگی قابل آزمایش و تعداد جواب ها زیاد باشد این الگوریتم بسیار مناسب میباشد.

تصادفی نمودن کنترل: مسئله ای مانند مرتب سازی را در نظر بگیرید که برای حل آن الگوریتم های زیادی وجود دارد. اگر این الگوریتم ها دارای زمان اجرای متوسط خوب و در بدترین حالت دارای زمان اجرای نامناسب باشند استفاده یکی از آنها برای حل همه مسائل پر مخاطره میباشد. برای حل این مشکل میتوان یکی از الگوریتم های فوق را بطور تصادفی انتخاب و اجرا نمود.

نمونه برداری تصادفی: در بسیاری از مسائل، با احتمال زیاد میتوان خصوصیات یک مجموعه را از روی خصوصیات یک زیر مجموعه تصادفی از آن مجموعه بدست آورد. در اینگونه موارد یک زیر مجموعه بصورت تصادفی انتخاب میشود و از روی خصوصیات این زیر مجموعه، خصوصیات مجموعه اصلی بدست میاید. الگوریتم نزدیکترین همسایه از این نمونه میباشد.

ازبین بردن تقارن: در بسیاری از مسائل محاسبات توزیع شده، فرایند ها باید به توافقی برسند که این توافق از طریق الگوریتم های قطعی امکان پذیر نیست. با تصادفی نمودن فرایندها، الگوهای یکسان (تقارن) در رفتار فرایندها شکسته میشود و در نتیجه فرایندها میتوانند به توافق مورد نظر برسند. دسترسی ایستگاههای کاری در شبکه های محلی به محیط ارتباطی یک نمونه از این مسائل میباشد.

۳- الگوریتم های تصادفی برای تشخیص تناظر دو گراف

در این بخش شش الگوریتم تصادفی از نوع لاس وگاس که از جستجوی تصادفی استفاده میکنند برای حل مسئله تشخیص تناظر دو گراف پیشنهاد شده است. در ابتدا به بررسی روشهای مختلف ساختن گرافهای تصادفی و چگونگی محاسبه اختلاف بین دو گراف میپردازیم و در ادامه به ارائه الگوریتم های پیشنهادی میپردازیم.

معمولا برای ساختن گراف تصادفی با تعداد گره های مشخص از یکی از دوروش زیر استفاده میشود. روش اول از یک عدد تصادفی برای مشخص نمودن وجود یا عدم وجود یک کمان در گراف استفاده میکند. روش دوم با یک تعداد کمان مشخص شروع میکند و بصورت تصادفی این کمانها را بین گره های مختلف قرار میدهد. در این مقاله از روش اول برای ساختن گرافها استفاده شده است.

یک گراف با وزن و جهت دار G را میتوان بصورت زوج $G = (V(G), W(G))$ نشان داد که در آن $V(G)$ مجموعه گره های گراف و $W(G)$ ماتریس همجواری گراف میباشد. درصورتیکه $|W_{ij}(G)| > 0$ باشد کمانی از گره i به گره j با وزن $W_{ij}(G)$ وجود دارد. گراف های بدون جهت ، دارای ماتریس همجواری متقارن میباشد. جایگشت های مختلف[2] سطرها (در نتیجه ستونهای متناظر) در ماتریس همجواری باعث تغییر ترتیب برچسب گره های گراف میگردد. بنابر این میتوان گفت دو گراف $G = (V(G), W(G))$ و $H = (V(H), W(H))$ متناظر هستند اگر وتنها اگر ترتیب برچسب گره های گراف H یک جایگشت از برچسب گره های گراف G باشد یعنی دو ماتریس همجواری $W(G)$ و$W(H)$ از طریق رابطه زیر بهم نگاشت داده میشوند[۸].

$$W(H) = P \,.\, W(G) \,.\, P^T \qquad (1)$$

1- Input Randomization
2- Permutation

که P ماتریس جایگشت میباشد. در هر سطر(ستون) این ماتریس تنها یک عدد ۱ وجود دارد و بقیه عناصر ماتریس صفر هستند. عضو P_{ji} از ماتریس P نشاندهنده نگاشت گره V_i از گراف G به گره V_j از گراف H میباشد. بر اساس مقدار ماتریس P، نگاشت $\sigma : V(G) \rightarrow V(H)$ بصورت زیر تعریف میگردد.

$$\sigma = \{(v_i, v_j) \mid P_{ji} = 1 ,\ v_i \in V(G),\ v_j \in V(H)\} \ \forall\, i, j = 1, 2, \ldots, n \tag{۲}$$

برای مثال اگر نگاشت σ بصورت $\sigma = \{(A, b), (B, d), (C, c), (D, a)\}$ باشد ماتریس جایگشت بصورت زیر خواهد شد.

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

با استفاده از نگاشت σ ، خطای نگاشت دو گراف G و H را بصورت زیر تعریف میکنیم.

$$J(\sigma) = \left\| W(H) - P \,.\, W(G) \,.\, P^T \right\| \tag{۳}$$

که ‖ . ‖ نرم ماتریس(هر نوع نرم) میباشد. بنابراین مسئله پیدا کردن تناظر بین دو گراف به حل معادله $J(\sigma) = 0$ تبدیل میشود. زمانیکه دو گراف بطور دقیق با هم متناظر نباشند (تناظر دو گراف با تصحیح خطا) مسئله پیدا کردن تناظر بین دو گراف به پیدا کردن کمینه تابع $J(\sigma)$ تبدیل میگردد. برای گراف با اندازه n ، محاسبه $J(\sigma)$ نیاز به زمانی از مرتبه $\theta(n^3)$ دارد. برای کم کردن هزینه اجرایی محاسبه $J(\sigma)$ ، میتوان نشان داد که تحت نگاشت σ، رابطه زیر برقرار است.

$$\left[P \,.\, W(G) \,.\, P^T\right]_{i,j} = [W(G)]_{\sigma(i),\, \sigma(j)} \tag{۴}$$

که $[A]_{ij}$ عضو سطر i و ستون j ماتریس A میباشد. اگر در رابطه (۳) از نرم L_1 ($\|W\| = \sum_i \sum_j |W_{i,j}|$) بجای نرم ‖ . ‖ استفاده کنیم شکل ساده تری برای محاسبه مقدار اختلاف بین دو گراف G و H بدست میاید. برای این منظور خطای نگاشت گره k از گراف G به گره σ(k) از گراف H بصورت زیر تعریف میشود.

$$J_k(\sigma) = \sum_{m=1}^{n} \left| [W(H)]_{k,m} - [W(G)]_{\sigma(k),\, \sigma(m)} \right| + \sum_{m=1}^{n} \left| [W(H)]_{m,k} - [W(G)]_{\sigma(m),\, \sigma(k)} \right| \tag{۵}$$

درصورتیکه گراف بدون جهت باشد خطای نگاشت گره k برابر است با

$$J_k(\sigma) = 2 \times \sum_{m=1}^{n} \left| [W(H)]_{k,m} - [W(G)]_{\sigma(k),\, \sigma(m)} \right| \tag{۶}$$

ودرنتیجه خطای تناظر دو گراف برای نگاشت σ بصورت زیر تعریف میگردد.

$$J(\sigma) = \sum_{k=1}^{n} J_k(\sigma) \tag{۷}$$

با توجه به معادلات فوق روشن است که محاسبه $J_k(\sigma)$ و $J(\sigma)$ نیاز به زمانی از مرتبه $\theta(n^2)$ دارد. در ادامه این بخش شش الگوریتم تصادفی برای تشخیص تناظر گراف آورده شده است. تمامی این الگوریتم ها از نوع الگوریتم های لاس وگاس میباشند با این تفاوت که احتمال همگرایی به جواب و متوسط زمان اجرای این الگوریتم ها متفاوت میباشند. این الگوریتم ها از جستجوی تصادفی برای پیدا نمودن تناظرگراف استفاده میکنند. رویه کلی این الگوریتم ها در شکل ۲ نشان داده شده است.

الگوریتم شماره ۱: در این الگوریتم، ابتدا نگاشت اولیه σ بصورت تصادفی بین گراف های G وH برقرار میگردد. سپس زوج $(u, \sigma(u))$ بصورت تصادفی انتخاب میشود بطوریکه $u \in V(G)$ و $\sigma(u) \in V(H)$ میباشد. زوج $(u, \sigma(u))$ بصورتی انتخاب میشود که برای عدد تصادفی k، مقدار $J_u(\sigma)$ ، kامین مقدار بزرگ را داشته باشد. اگر مقدار خطای حاصل از نگاشت $(J_u(\sigma))$ بزرگتر از مقدار آستانه (مقدار آستانه بصورت تطبیقی مشخص میگردد و مقدار آن در هر لحظه برابر است با میانگین خطای نگاشت برای تمامی گره ها) باشد گره u با گره v جابجا میشود. گره v بصورتی انتخاب میگردد که برای یک عدد تصادفی m ، $J_v(\sigma)$ ، mامین مقدار بزرگ داشته باشد. یعنی در نگاشت جدید (نگاشت 'σ) دو زوج $(u, \sigma(u))$ و $(v, \sigma(v))$ به دو زوج $(u, \sigma(v))$ و $(v, \sigma(u))$ تبدیل شده و زوج های دیگر بدون تغییر میمانند. این الگوریتم در شکل ۳ نشان داده شده است.

این الگوریتم برای گراف های کوچک مناسب میباشد ولی اگر اندازه گراف بزرگ شود این الگوریتم کارایی خود را از دست میدهد زیرا سطح تابع $J(\sigma)$ مملو از پستی و بلندی است و ممکن است الگوریتم مدت زیادی در حداقل های محلی این تابع گرفتار شود. دلیل دیگر افزایش زمان اجرا الگوریتم، تصمیم گیریهای کورکورانه میباشد که باعث میشود تا جابجایی های غلط زیادی صورت گیرد.

تابع Select (k)، برچسب گره ای (u) را بعنوان جواب برمیگرداند که مقدار خطای نگاشت این گره $(J_u(\sigma))$ kامین مقدار بزرگ را در مجموعه خطاهای نگاشت ها دارا باشد. در بیشتر موارد، اگر یک الگوریتم تصادفی بصورت بازگشتی صدا زده شود. همراه با کاهش اندازه مسئله، احتمال همگرایی الگوریتم کاهش میابد. یک راه خوب ، قطع عمل صدا زدن بازگشتی الگوریتم تصادفی و استفاده از یک الگوریتم قطعی بجای الگوریتم تصادفی است. در الگوریتم های ارائه شده در این مقاله، قطع عمل صدا زدن بازگشتی در انتخاب گره u انجام شده است. در این تابع اگر خطا از مقدار مشخصی کمتر شود بجای انتخاب تصادفی یک گره، گرهی انتخاب میگردد که بیشترین خطا را تولید نموده باشد.

الگوریتم شماره ۲: این الگوریتم مانند الگوریتم ۱ از جستجوی کورکورانه برای پیدا نمودن نگاشت استفاده میکند و دارای مشکلاتی همانند مشکلات الگوریتم ۱ میباشد با این تفاوت که بجای انتخاب گره v با mامین مقدار بزرگ $J_v(\sigma)$، این گره بصورتی انتخاب میگردد که $J_v(\sigma)$ دارای

بزرگترین مقدارباشد. جابجایی گره u با گرهی که دارای بیشترین مقدار خطاست باعث کاهش بیشتر خطا در مقایسه بـا الگوریتـم ۱ میگـردد و ایـن باعث میگردد که زمان متوسط اجرای این الگوریتم کاهش یابد. شکل ۴ الگوریتم ۲ را برای تناظر گراف نشان میدهد.

```
function GraphIsomorphisms (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         Compute J_m(σ)  // Compute the error for each node
      end for
      Select a pair (u ∈ G, v ∈ H) according to some criteria
      Reorder the mapping inorder to find global minimum of J (σ)
   until ( J (σ) = 0)
   return σ
end GraphIsomorphism
```

شکل ۲: الگوریتم تصادفی برای تناظر دو گراف

```
function GraphIsomorphism (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k), σ(m)}|
      end for
      u = Select (random (N))    // random number in range [1, N]
      if J_u(σ) > T then
         U = Select (random (N)) // random number in range [1, N]
         Swap (u, U)
      end if
   until J (σ) = 0
   return σ
end GraphIsomorphism
```

شکل ۳: الگوریتم ۱ برای تناظر دو گراف

الگوریتم شماره ۳: این الگوریتم مانند الگوریتم ۲ میباشد با این تفاوت که اگر جابجایی دو گره u و v نسبت به گذشته خطای کمتری را تولید نماید این دو گره جابجا میگردند و در غیر اینصورت این نگاشت بهمان صورت گذشته باقی میماند. تعداد جابجایی های غلط این الگوریتم بسیار کمتر از الگوریتم ۲ میباشد و بهمین دلیل دارای متوسط زمان اجرای کمتری نسبت به الگوریتم ۲ میباشد. الگوریتم ۳ برای تناظر گراف در شکل ۵ آورده شده است.

الگوریتم شماره ۴: در این الگوریتم گره u از گراف G بصورتی انتخاب میگردد که خطای حاصل از نگاشت این گره، kامین مقدار بزرگ را داشته باشد. سپس گره v بصورت تصادفی از گراف H انتخاب میگردد. در صورتیکه تحت نگاشت σ، گره های u و v با هم متناظر نباشند و خطای نگاشت حاصل از جابجاییu و v کمتر از مقدار آستانه ای و همچنین کمتر از خطای نگاشت σ باشد این دو گره جابجا میشوند. اگر تحت نگاشت σ، گره های u و v با هم متناظر باشند و خطای نگاشت حاصل از جابجاییu و v بیشتر از مقدار آستانه ای باشد گره u با گرهی جابجا میگردد که کمترین خطا را تولید نماید. این الگوریتم در شکل ۶ آورده شده است. متوسط زمان اجرای این الگوریتم برای گراف های کوچک مناسب میباشد اما با بزرگ شدن اندازه گرافها، تعداد انتخاب ها زیاد میشوند و باعث زیاد شدن متوسط زمان اجرای الگوریتم میگردد. اما بدلیل اینکه نسبت به جستجوی کورکورانه الگوریتم های ۱ و ۲ تصمیم گیریهای مناسب تری را انجام میدهد از دو الگوریتم ۱ و ۲ بهتر است.

```
function GraphIsomorphisms (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k), σ(m)}|
      end for
      u = Select (random (N)) // random number in range [1, N]
      if J_u(σ) > T then
         MoveNode (u)
      end if
   until J (σ) = 0
   return σ
end GraphIsomorphism

procedure MoveNode (u)
   U = Select (1)    // The largest element
   Swap (u, U)
end MoveNode
```

شکل ۴: الگوریتم ۲ برای تناظر دو گراف

```
function GraphIsomorphisms (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k), σ(m)}|
      end for
      u = Select (random (N))    // random number in range [1, N]
      if J_u(σ) > T then
         MoveNode (u)
      end if
   until J (σ) = 0
   return σ
end GraphIsomorphism

procedure MoveNode (u)
   U = Select (1)
   Create σ' from σ by sawpping u and U
   if J_u(σ) ≥ J_u(σ') then
      Swap (u, U)
   end if
end MoveNode
```

شکل ۵: الگوریتم ۳ برای تناظر دو گراف

الگوریتم شماره ۵: این الگوریتم مانند الگوریتم ۴ میباشد اما بجای انتخاب تصادفی گره v از بین همه گره های گراف H ، گرهی انتخاب میگردد که خطای آن در نگاشت σ، kامین مقدار بزرگ را داشته باشد. بنابراین نسبت به الگوریتم ۴ (کـه تعداد زیادی از انتخابهایش شرایط جابجایی را

ندارند) از تعداد انتخاب های کمتری برای پیدا نمودن جواب استفاده میکنند. بنابراین دارای متوسط زمان اجرای کمتری نسبت به الگوریتم ۴ میباشد. شکل ۷ این الگوریتم را نشان میدهد.

```
function GraphIsomorphism (G, H)
  Create a random mapping σ
  repeat
    for m = 1 to n do
      J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k),σ(m)}|
    end for
    u = Select (random (N)) // random number in range [1, N]
    v = σ (random (n))
    Create σ' from σ by sawpping u and σ^-1(v)
    if J_u(σ') < T then
      if σ(u) <> v and J_u(σ') < J_u(σ) then
        Swap (u, σ^-1(v))
      end if
    else
      if σ(u) = v then
        MoveNode (u)
      end if
    end if
  until J (σ) = 0
end GraphIsomorphism

procedure MoveNode (u)
  BestError = J_u(σ)
  BestNode = u
  for U = 1 to n do
    Create σ' from σ by sawpping u and U
    if J_u(σ') < BestError then
      BestError = J_u (σ')
      BestNode = U
    end if
  end for
  Swap (u, U)
end MoveNode
```

شکل ۶: الگوریتم ۴ برای تناظر دو گراف

```
function GraphIsomorphism (G, H)
  Create a random mapping σ
  repeat
    for m = 1 to n do
      J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k),σ(m)}|  end for
    end for
    u = Select (random (N)) // random number in range [1, N]
    v = σ (Select (random (N)))
    Create σ' from σ by sawpping u and σ^-1(v)
    if J_u(σ') < T then
      if s(u) <> v then
        Swap (u, σ^-1(v))
      end if
    else
      if s(u) = v then
        MoveNode (u)
      end if
    end if
  until J (σ) = 0
end GraphIsomorphism

procedure MoveNode (u)
  BestError = Ju(σ)
  BestNode = u
  for U = 1 to n do
    Create σ' from σ by sawpping u and U
    if J_U(σ') < BestError then
      BestError = J_U (σ')
      BestNode = U
    end if
  end for
  Swap (u, U)
end MoveNode
```

شکل ۷: الگوریتم ۵ برای تناظر دو گراف

الگوریتم شماره ۶: این الگوریتم تلفیقی از الگوریتم های ۳و ۵ میباشد و بدین وسیله احتمال انتخاب گره صحیح u افزایش میابد. در این الگوریتم یکی از گره هایی که بیشترین خطا را در نگاشت σ تولید نموده است انتخاب میگردد واین گره با احتمال زیاد گره مناسب خود را پیدا میکند. بنابراین نسبت به دو الگوریتم ۳و ۵ از متوسط زمان اجرای کمتری برخوردار است. این الگوریتم در ۸ نشان داده شده است.

الگوریتمهای تناظر دو گراف با تصحیح خطا: همه الگوریتم های فوق را میتوان برای گراف های دارای اغتشاش بکار برد تنها با این تفاوت که شرط پایان را باید بصورتی مشخص نماییم که زمان اجرای الگوریتم پس از پیدا نمودن تناظر بهینه محدود باشد. با توجه به اینکه در الگوریتم های تصادفی احتمال گرفتاری در حداقل های محلی بسیار کم (و در بیشتر موارد غیر ممکن) میباشد(زیرا از اطلاعات محلی برای پیدا نمودن جواب بهینه استفاده نمیکند) بنابر این اگر مدت زیادی الگوریتم نتواند جواب را تغییر دهد این جواب بمنزله جواب بهینه است. این مدت برای الگوریتم های فوق متفاوت میباشد. برای نمونه در شکل ۹ الگوریتم ۶ بگونه ای اصلاح گردیده است که علاوه بر گرافهای بدون اغتشاش، برای گراف های دارای اغتشاش نیز قابلیت اجرا داشته باشد.

۴- نتایج شبیه سازیها

در این بخش الگوریتم های فوق روی گراف های مختلف و با اندازه های متفاوت آزمایش شده اند و نتایج آن درجداول زیر آمده است. در این مقاله از روش اول برای ساختن گراف تصادفی باوزن و بدون جهت G استفاده شده است که در آن اعداد تصادفی در فاصله [۰، ۹۹] بعنوان وزن هر کمان اختصاص داده میشود و وزن صفر بمعنای عدم وجود کمان میباشد. با تغییر تصادفی برچسب n گره از گراف G، گراف H تولید میگردد. در تمامی آزمایشهای زیرهر الگوریتم روی ۱۰ گراف یکسان اجرا شده است تا توزیع زمان اجرای الگوریتم ها مشخص گردد و مقدار متوسط تکرار حلقه repeat-until و تعداد اجراهایی که همگرا نشده اند در جدول ۱ آورده شده است. در این آزمایشها، مقدار آستانه بصورت تطبیقی مشخص میگردد و مقدار آن در هر لحظه برابر است با میانگین خطای نگاشت برای تمامی گره ها. برای تولید گرافهای دارای اغتشاش، وزن هر کمان را با عددی تصادفی در فاصله [-X, X] جمع میشود که X درصد اغتشاش میباشد.

```
function GraphIsomorphism (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k), σ(m)}|
      end for
      u = Select (random (N)) // random number in range [1, N]
      if J_u(σ) > T then
           MoveNode (u)
      end if
   until J (σ) = 0
end GraphIsomorphism
procedure MoveNode (u)
   BestError = Ju(σ)
   BestNode = u
   for U = 1 to n do
      σ'  is new mapping, which drived from σ by sawpping u and
      U
      if J(σ') <  BestError then
         BestError = J_U (σ')
         BestNode = U
      end if
   end for
   Swap (u, U)
end MoveNode
```

شکل ۸: الگوریتم ۶ برای تناظر دو گراف

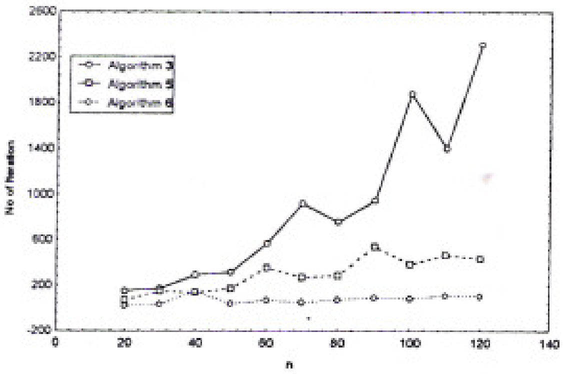
```
function ErrorCorrectingGraphIsomorphism (G, H)
   Create a random mapping σ
   repeat
      for m = 1 to n do
         J_k(σ) = 2 × Σ_{m=1}^{n} |W(H)_{k,m} - W(G)_{σ(k), σ(m)}|
      end for
      u = Select (random (N)) // random number in range [1, N]
      if J_u(σ) > T then
           MoveNode (u)
      end if
   until J (σ) = 0 or for  long period  J (σ) is constant
end ErrorCorrectingGraphIsomorphism
procedure MoveNode (u)
   BestError = Ju(σ)
   BestNode = u
   for U = 1 to n do
      Create σ' from σ by sawpping u and U
      if J_U(σ') <  BestError then
         BestError = J_U (σ')
         BestNode = U
      end if
   end for
   Swap (u, U)
end MoveNode
```

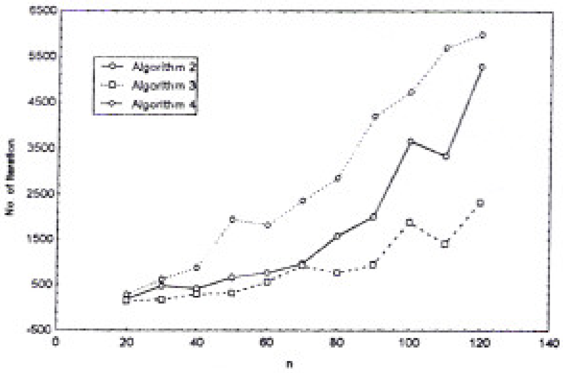
شکل ۹: الگوریتم تناظر دو گراف با تصحیح خطا

جدول ۱: متوسط تعداد تکرارحلقه repeat-until الگوریتم های تصادفی تناظر دو گراف برای ده اجرای مختلف

	متوسط تکرار الگوریتم						تعداد اجراهای همگرا نشده					
الگوریتم / اندازه گراف	۱	۲	۳	۴	۵	۶	۱	۲	۳	۴	۵	۶
۲۰	۴۲۴	۱۸۹	۱۵۱	۲۸۷	۶۵	۱۸	۰	۱	۰	۰	۰	۰
۳۰	۲۳۸	۴۷۹	۱۷۳	۶۲۰	۱۵۱	۳۲	۰	۰	۰	۰	۰	۰
۴۰	۵۵۴	۴۱۹	۲۹۲	۸۷۳	۱۳۵	۱۴۵	۰	۰	۱	۰	۰	۰
۵۰	۵۹۳	۶۶۶	۳۱۵	۱۹۲۴	۱۷۱	۳۷	۰	۰	۰	۰	۰	۰
۶۰	۱۲۸۰	۷۵۹	۵۶۸	۱۸۱۲	۳۵۳	۶۹	۰	۰	۰	۰	۰	۰
۷۰	۲۹۷۶	۹۶۴	۹۲۳	۲۳۵۵	۲۶۸	۴۹	۰	۰	۰	۰	۰	۰
۸۰	۳۵۲۳	۱۵۸۱	۷۶۰	۲۸۶۱	۲۸۶	۷۴	۰		۰	۰	۰	۰
۹۰	۴۴۷۷	۲۰۰۷	۹۴۸	۴۲۰۰	۵۴۴	۹۱	۰	۰۰	۰	۱	۰	۰
۱۰۰	●	۳۶۵۶	۱۸۸۱	۴۷۳۱	۳۸۵	۸۴	۱۰	۰	۰	۰	۰	۰
۱۱۰	●	۳۳۴۰	۱۴۰۶	۵۷۱۲	۴۶۵	۱۰۹	۱۰	۰	۳	۰	۰	۰
۱۲۰	●	۵۲۸۰	۲۳۱۵	۵۹۹۱	۴۲۶	۱۰۵	۱۰	۰	۰	۰	۰	۰

الگوریتم ۱ برای گراف های بزرگتر از ۱۰۰ گره در ۱۰۰۰۰ تکرار حلقه همگرا نمیشود. شکل ۱۰ متوسط تعداد تکرار های حلقه repeat-until را برای الگوریتم های ۲ تا ۶ نشان میدهد.





شکل ۱۰: متوسط تعداد تکرار های حلقه repeat-until

با توجه به شکل ۱۰ میتوان گفت که متوسط زمان اجرای الگوریتم های ۱ تا ۴ از مرتبه $O(n^4)$ و متوسط زمان اجرای الگوریتم های ۵ و ۶ از مرتبه $O(n^3)$ میباشند. جدول ۲ متوسط تعداد تکرار حلقه repeat-until و تعداد اجرا های همگرا نشده و خطا پس از همگرایی الگوریتم تناظر دو گراف با تصحیح خطا برای در صد اغتشاش های مختلف را نشان میدهند. برای جزئیات بیشتردرباره شبیه سازیها به مرجع [۱۴] مراجعه شود.

جدول ۲: متوسط تعداد تکرارحلقه repeat-until الگوریتم تناظر دو گراف با تصحیح خطا برای اغتشاشهای مختلف

اغتشاش	اندازه گراف	۲۰	۳۰	۴۰	۵۰	۶۰	۷۰	۸۰	۹۰	۱۰۰	۱۱۰	۱۲۰
٪۵	متوسط تکرار الگوریتم	۱۱۶	۴۳۲	۶۷	۱۳۸	۲۲۹	۸۵	۱۱۵	۱۲۲	۱۲۵	۱۱۲	۱۵۴
	اجراهای همگرا نشده	۰	۰	۱	۰	۰	۰	۰	۰	۰	۰	۰
	خطا پس از همگرایی	۱۷۹۶	۴۵۱۶	۷۶۲۰	۱۲۱۵۶	۱۷۲۳۲	۲۳۷۳۶	۳۰۹۶۰	۳۹۶۶۰	۴۸۳۹۸	۵۹۲۴۶	۷۰۳۷۶
٪۱۰	متوسط تکرار الگوریتم	۱۶	۶۵	۲۳	۳۸	۵۶	۱۲۲	۹۲	۷۱	۱۲۲	۶۷	۱۲۰
	اجراهای همگرا نشده	۰	۰	۰	۰	۰	۰	۰	۰	۱	۰	۰
	خطا پس از همگرایی	۳۷۱۲	۸۹۳۸	۱۵۱۰۴	۲۴۰۰۶	۳۴۴۶۴	۴۷۵۵۴	۶۱۷۹۸	۷۸۲۹۸	۹۸۲۲۴	۱۱۷۰۴۰	۱۳۹۱۴۶
٪۱۵	متوسط تکرار الگوریتم	۹۳	۱۶۰	۶۴	۷۱	۱۱۹	۱۰۷	۱۰۷	۸۹	۱۵۶	۲۰۴	۱۳۱
	اجراهای همگرا نشده	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰
	خطا پس از همگرایی	۵۵۳۸	۱۲۸۵۶	۲۲۹۴۶	۳۵۹۴۰	۵۲۱۲۴	۷۰۰۰۰	۹۱۷۷۶	۱۱۵۷۷۰	۱۴۴۵۹۶	۱۷۵۹۲۸	۲۰۸۱۳۲

۵- نتیجه گیری

در این مقاله شش الگوریتم تصادفی برای مسئله تناظر دو گراف ارائه شده است. آزمایشهای مختلف نشان میدهند که برای گراف با اندازه n، بعضی از این الگوریتم ها دارای پیچیدگی از مرتبه $O(n^4)$ و بعضی دیگر دارای پیچیدگی از مرتبه $O(n^3)$ میباشند. با توجه به اینکه الگوریتم Backtracking دارای پیچیدگی زمانی از مرتبه $O(n!)$ میباشد الگوریتم های پیشنهاد شده در مقایسه با الگوریتم Backtracking بهبود قابل ملاحظه ای را در مرتبه بزرگی نشان میدهند. زمانی که گراف ها دارای اغتشاش باشند مسئله بسیار پیچیده تر میباشد ولی شبیه سازیها نشان میدهند که مرتبه بزرگی الگوریتم تصادفی ۶ برای گراف های با اغتشاش و بدون اغتشاش یکسان است.

۶- مراجع

[1] Abadir, M.S. and Ferguson, J., "An Improved Layout Verification Algorithm", Proc. of IEEE European Int. Conf. on Design and Automation, pp. 391-395, 1990.

[2] Maurer, P.M. and Schapira, A.D., "A Logic-To-Logic Comparator for VLSAI Layout Verification", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7, No. 8, pp. 897-907, 1988.

[3] Huang, K.-T.and Overhauser, D., "A Novel Graph Algorithm for Circuit Recognition", Proc. of IEEE Int. Symposium on Circuits and Systems, pp. 1695-1698, 1995.

[4] Petrank, E.and Roth, R.M., "Is Code Equivalence easy to decide", IEEE Trans. on Information Theory, Vol. 43, No. 5, pp. 1602-1604, 1997.

[5] Schalkoff, R. J., Pattern Recognition: Statistical, Structural and Neural Approaches, John Whiley, New York, 1992.

[6] Cinque, L., Yasuda, D., Shapiro, -L. G., Tanimoto, S., and Allen, B., "An Improved Algorithm for Relational Distance Graph Matching", Pattern Recognition, Vol. 29, No. 2, pp. 349-359, 1996.

[7] Wilson, R. C. and Hancock, E. R., "Structural Matching by Discrete Relaxation", IEEE Trans. on Pattern Analysis and Machine Vision, Vol. 19, No. 6, pp. 634-648, 1997.

[8] Wang, Y., Fan, K., and Horng, J., "Genetic-Based Search for Error-Correcting Graph Isomorphism", IEEE Trans. on Systems, Man, and Cybernetics-PartB: Cybernetics, Vol. 27, No. 4, pp. 588-597, 1997.

[9] Depiero, F. Trived, M., and Serbin, S., "Graph Matching Using Direct Classification of Node Attendance", Pattern Recognition, Vol. 29, No. 6, pp. 1031-1048, 1996.

[10] Agusa, K., Fujita, S., Yamashita, M., and Ae, T., "On Neural Networks for Graph Isomorphism Problem", Proc. of RNNS/IEEE Int. Symposium on Neuroinformatics and Neurocomputers, pp. 1142-1148, 1992.

[11] Krcmar, M. and Dhawan, A. P., "Application of Genetic Algorithms in Graph Matching", Proc. of IEEE Int. Conf. on Conference on Neural Networks (ICNN'94), pp. 3872-3876, 1994.

[12] Gupta, R., Smolka, S. A., and Bhaskar, S., "On Randomization in Sequential and Distributed Algorithms", ACM Computing Surveys, Vol. 26, No. 1, pp. 7-85, 1994.

[13] Brassard, G. and Bratley, P., Algorithmics: Theory and Practice, Printce-Hall, 1988.

[14] Beigy, H. and Meybodi, M. R., "Randomized Las Vegas Algorithms for Graph Isomorphism", Technical Report, Computer Eng. Dept., Amirkabir University of Technology, Tehran, Iran , 1998.

هفتمین کنفرانس مهندسی برق ایران

مرکز تحقیقات مخابرات ایران ۲۷ - ۲۹ اردیبهشت ماه ۱۳۷۸

ITRC

مجموعه مقالات

کامپیوتر