علوم رایانش و فناوری اطلاعات
نشریه علمی انجمن کامپیوتر ایران
مجلد ۱۷، شماره ۲، ۱۳۹۸
صفحات ۵۱- ۴۵
مقاله عادی

الگوریتمی ابتکاری جهت مسئله تشخیص یکریختی گراف‌ها مبتنی بر دوری از مرکز گراف

علی نوراله[۱]، سمیه چک[۲*]

*نویسنده مسئول، دریافت: ۹۸/۱۲/۱۱، بازنگری: ۹۹/۰۲/۱۶، پذیرش: ۹۹/۰۳/۲۳

[۱] استادیار، مهندسی کامپیوتر، دانشکده مهندسی کامپیوتر، دانشگاه تربیت دبیر شهید رجایی، تهران، ایران
[۲] دانشجوی کارشناسی ارشد، مهندسی کامپیوتر ، دانشکده مهندسی کامپیوتر، دانشگاه تربیت دبیر شهید رجایی، تهران، ایران

چکیده

مسئله یکریختی گراف‌ها از مجموعه مسائل باز ازلحاظ پیچیدگی محاسباتی است که فقط تعلق آن به کلاس NP مشخص است ولی تعلق آن به P یا NP-Complete مشخص نیست. راه‌حل مسئله در زمان چندجمله‌ای هنوز ناشناخته است و لذا زمینه برای تحقیق و ایده‌پردازی فراهم می‌باشد. ازاین‌رو الگوریتم‌های چندجمله‌ای برای این مسئله جزو الگوریتم‌های ابتکاری[۱] محسوب می‌شوند. این مقاله به بررسی راه‌های تعیین یکریختی دو گراف متناهی با یکدیگر و ارائه یک روش ابتکاری جدید می‌پردازد. الگوریتمی پیشنهاد می‌شود که گراف ورودی را به یک رشته‌کد پرانتزی تبدیل می‌کند و سپس به‌جای مقایسه دو گراف رشته‌کدهای آن دو گراف باهم مقایسه می‌شوند و یکریختی یا عدم یکریختی میان آن‌ها تشخیص داده می‌شود. زمان اجرای این الگوریتم $O(ne)$ می‌باشد که در آن n تعداد رأس‌های گراف و e تعداد یال‌های گراف است و در دسته الگوریتم‌های "برچسب‌گذاری کانونی[۲]" برای گراف‌های "ساده[۳]، همبند[۴] و بدون برچسب[۵]" قرار دارد. بعد از پیاده‌سازی این الگوریتم و بررسی نتایج آن مشخص شد که با عملکرد صحیح بیشتر از ۹۹٪، عدم یکریختی میان گراف‌های غیریکریخت[۶] به‌درستی تشخیص داده می‌شود.

کلمات کلیدی: یکریختی گراف، الگوریتم چندجمله‌ای، برچسب‌گذاری کانونی، گراف ساده همبند، الگوریتم ابتکاری

۱- مقدمه

گراف‌ها ساختمان داده‌ای قدرتمند برای نمایش اشیا و مفاهیم آن‌ها هستند. گراف یک ساختمان داده غیرخطی است که شامل دو مجموعه می‌باشد؛ مجموعه غیر تهی از رأس‌ها و مجموعه‌ای از یال‌ها که رأس‌ها را به هم متصل می‌کنند. با ایجاد ترکیبات مختلف از گره‌ها و یال‌های متصل به آن‌ها به شکل‌های متفاوتی از گراف دست می‌یابیم. برخی از مسائل دنیای واقعی را می‌توان به کمک گراف‌ها نمایش داد و با استفاده از رأس‌ها و یال‌های آن ارتباط بین اشیا را توصیف کرد. به‌وسیله گراف می‌توان مسئله را به‌خوبی شبیه‌سازی کرد و میزان درک از مسائل را افزایش داد. امروزه اطلاعات مربوط به شبکه‌های اجتماعی، پایگاه داده‌ها، پیوندهای شیمیایی و زیست‌شناسی نیاز به استفاده از ساختار بزرگی دارند که به‌طورمعمول توسط گراف‌ها نمایش داده می‌شوند[۱-۴]. علاوه بر این، داده‌های تولید شده که حجم آن‌ها به‌مرورزمان در حال افزایش است به تجزیه‌وتحلیل نیاز دارند؛ نمونه‌های قابل توجه ساختارهای مولکولی و پروتئین، شبکه‌های تعامل و اخیراً ژنوم که چندین سال است به‌عنوان رشته‌ای از پایه‌ها ارائه می‌شود[۵]، از طرفی تعامل پروتئین با پروتئین، تعامل DNA و پروتئین، تعامل متابولیک، اتصال فاکتور رونویسی، شبکه‌های عصبی از این طریق قابل درک هستند[۶]. بسیاری از ساختارهای شیمیایی و بیولوژیکی به‌راحتی به‌صورت گراف‌ها قابل مدل‌سازی هستند[۷]، برای مثال یک ساختار مولکولی را می‌توان یک گراف دانست که گره‌های آن با اتم‌ها و یال‌های آن با پیوندهای شیمیایی مطابقت دارند. در سال‌های اخیر، انجمن علمی فعال در زمینه‌های تحلیل الگو، تشخیص الگو و بینایی کامپیوتر[۸] به مبحث گراف‌ها توجه بیشتری نشان داده و به دنبال آن برنامه‌های کاربردی مبتنی بر گراف‌ها چند برابر شده‌اند.

در تئوری گراف دو گراف یکریخت گراف‌هایی هستند که ازلحاظ ساختاری باهم یکسان باشند به‌بیان‌دیگر الگوی اتصال میان گره‌ها در هر دو گراف باهم برابر باشد.

مسئله یکریختی گراف ازجمله مسائل تصمیم‌گیری است که هدف آن تعیین یکسان بودن یا نبودن دو گراف متناهی ازلحاظ توپولوژی ارتباطی است. این مسئله از معدود مسائل الگوریتمی است که هم برای آن الگوریتم چندجمله‌ای یافت نشده و هم اینکه ثابت نشده جزو مسائل NP-Complete است، ازاین‌رو در مجموعه مسائل NP قرار دارد[۹].

تشخیص یکریختی در زمینه‌های پردازش تصویر، شبکه‌های اجتماعی، ساختار پیوندهای شیمیایی، تقسیم‌بندی ژن‌ها، تجزیه‌وتحلیل پروتئین، زیست‌شناسی سلولی- مولکولی و نگاشت الگوریتم‌های موازی که برای یک توپولوژی خاص طراحی شده‌اند به یک توپولوژی دیگر، نقش برجسته‌ای دارد[۱۰-۱۲].

الگوریتم‌های پیشنهادی از چندین دهه پیش و با روش‌های متفاوت مطرح شده‌اند و با توجه به باز بودن مسئله، روند ارائه پیشنهادها ادامه دارد. نقاط قوت الگوریتم پیشنهادی در تعیین پیچیدگی و مجموعه گراف ورودی می‌باشد؛ پیچیدگی زمانی الگوریتم‌های موجود بر اساس زمان اندازه‌گیری شده در حین اجرای الگوریتم و با ورودی گراف‌های تصادفی تعیین شده است درحالی‌که الگوریتم پیشنهادی دارای پیچیدگی قطعی ازنظر زمان و فضا می‌باشد. همچنین میزان صحت عملکرد الگوریتم‌های موجود بر اساس مجموعه بزرگی از گراف‌های مشخص شده از سوی طراحان اعلام شده است که در الگوریتم پیشنهادی در این مقاله صحت عملکرد بر روی کلیه گراف‌های موجود با اندازه کوچک و زیرمجموعه‌ای از گراف‌های با اندازه بزرگ‌تر بیان می‌شود.

الگوریتم ابتکاری در این مقاله برای حل مسئله یکریختی گراف‌های بدون برچسب پیشنهاد می‌شود که هدف آن کاهش پیچیدگی زمانی و ارائه یک الگوریتم چندجمله‌ای با بهبود نتایج به‌دست‌آمده است. تعیین یکریختی در گراف‌های بدون برچسب فرآیندی بسیار دشوار و دارای پیچیدگی زمانی بالا است زیرا گره‌ها برای تشخیص هویت خود از گره‌های دیگر، دارای وزن نیستند. الگوریتم پیشنهادی برای اجرا بر روی گراف‌های بدون برچسب تصادفی طراحی شده است. در ادامه این مقاله پس از ارائه تعاریف مورد نیاز در بخش ۲، در بخش ۳ به بررسی کارهای مرتبط پرداخته می‌شود. بخش ۴ مقاله حاوی الگوریتم پیشنهادی و بخش ۵ شامل ارزیابی عملکرد آن است و در آخر در بخش ۶ نتایج این پژوهش ذکر خواهد شد.

۲- تعاریف مورد نیاز

یکریختی در نظریه گراف‌ها، یک نگاشت دوسویه میان مجموعه رأس‌های دو گراف است؛ به‌طوری‌که هر دو رأس در گراف اول مجاور هستند اگر و فقط اگر نگاشت آن دو رأس در گراف دوم هم مجاور باشند. این نگاشت دوسویه (یک‌به‌یک و پوشا) میان رأس‌ها، با حفظ نگاشت میان یال‌های دو گراف نیز توصیف می‌شود. بنابراین یکریختی، نگاشت دوسویه میان ساختار دو گراف است.

فرض کنید $G = (V, E)$ و $G' = (V', E')$ گراف‌هایی به ترتیب با مجموعه رأس‌های V و V' و مجموعه یال‌های E و E' باشند، G و G' یکریخت هستند اگر و تنها اگر نگاشت یک به یک و پوشا $f: V \to V'$ و $g: E \to E'$ وجود داشته باشند به‌طوری‌که به ازای هر $e = (v_i, v_j)$ که در آن $e \in E$، بتوان نتیجه گرفت: $g(e) = \big(f(v_i), f(v_j)\big)$ که در آن $g(e) \in E'$.

به بیان ساده تشخیص یکریختی میان دو گراف به این معناست که آیا دو گراف، ساختار توپولوژیک یکسان دارند؟ در صورت وجود یکریختی میان دو گراف، آن دو گراف را یکریخت می‌گویند و آن را با عبارت $G \cong G'$ نشان می‌دهند.

گرافی که در آن بین هر دو رأس آن یک مسیر وجود داشته باشد را گراف همبند می‌نامند. اگر فاصله هر دو رأس مجاور در یک گراف عدد ۱ در نظر گرفته شود آنگاه برای هر دو رأس u و v از گراف کوتاه‌ترین فاصله بین این دو رأس با $dist(u, v)$ نشان داده می‌شود. در یک گراف همبند $G = (V, E)$ مقدار خروج از مرکز برای هر رأس $v \in V$ برابر با فاصله‌ی دورترین رأس در گراف نسبت به رأس v تعریف می‌شود که با عبارت $ecc(v)$ [۷] نمایش می‌دهند. به بیان دیگر

$$\forall v \in V: ecc(v) = \max_{u \in V}\{dist(v, u)\}.$$

قطر یک گراف که با $diam(G)$ نشان داده می‌شود برابر با بیشترین مقدار ecc روی همه رأس‌های آن گراف است و شعاع یک گراف که با $radius(G)$ نشان داده می‌شود برابر با کمترین مقدار ecc روی همه رأس‌های آن گراف است. به بیان دیگر

$$diam(G) = \max_{u,v \in V}\{dist(u, v)\},$$
$$radius(G) = \min_{u,v \in V}\{dist(u, v)\}.$$

مجموعه رئوسی که بیشترین ecc را دارند محیط گراف و مجموعه رئوسی که کمترین ecc را دارند مرکز گراف نامیده می‌شوند و به ترتیب با $Circumference(G)$ و $Centre(G)$ نشان داده می‌شوند. به بیان دیگر

$$Circumference(G) = \{v: ecc(v) = diam(G)\},$$
$$Centre(G) = \{v: ecc(v) = radius(G)\}.$$

گراف‌هایی که در آن همه رئوس دارای مقدار ecc یکسان باشند گراف‌های خودمرکز[۸] نامیده می‌شوند[۱۳].

در مسئله یکریختی گراف‌ها، گروهی از الگوریتم‌ها بر اساس برچسب‌گذاری کانونی ارائه شده‌اند. برچسب زدن کانونی یک گراف به معنای اختصاص یک برچسب منحصر به فرد به هر رأس است به گونه‌ای که دو گراف یکریخت هستند اگر و فقط اگر برچسب به‌دست‌آمده از دو گراف با یکدیگر همخوانی داشته باشد. در ادامه دو دسته اصلی از روش‌های برچسب زدن کانونی[۱۴] را شرح داده می‌شود.

در دسته‌بندی اول از یک تابع با ورودی رأس یا گراف استفاده کرده که یک مقدار مشخص (برچسب کانونی) به ترتیب برای رأس یا گراف مربوطه خروجی می‌دهد. برای مثال می‌توان یک تابع کدگذاری کانونی دودویی بر روی رأس‌های گراف ایجاد کرد.

در دسته‌بندی دوم از یک مقدار ثابت برای مرتب‌سازی لغوی برچسب تمامی رأس‌های یک گراف استفاده می‌کند. برای مثال لیست درجه مربوط به همسایگان یک رأس را در نظر گرفته و سپس به‌عنوان برچسب کانونی رأس بر روی آن مرتب‌سازی انجام می‌دهد. در روشی دیگر از خصوصیت دنباله فاصله‌ای هر رأس به‌عنوان برچسب رأس استفاده می‌کند. الگوریتم پیشنهادی در این مقاله جزو دسته‌بندی دوم محسوب می‌شود.

۳- کارهای مرتبط

مسئله یکریختی گراف‌ها جزو مسائل باز است که اولین بار توسط Karp در چهار دهه قبل به‌عنوان یک مسئله مهم به آن اشاره شده است[۹]. اگر تعداد رأس‌ها در هر گراف n باشد برای تعیین دو رأس متناظر میان دو گراف، لازم است یک رأس از گراف اول با n رأس از گراف دوم مقایسه شود و سپس رأس دیگری از گراف اول با $n-1$ رأس دیگر قابل تناظر است و این روند نشان می‌دهد زمان اجرای یک جستجوی جامع برابر $O(n!)$ است. هیچ الگوریتم چندجمله‌ای برای یکریختی گراف شناخته نشده است و حدس زده شده که هیچ الگوریتم چندجمله‌ای نیز نمی‌تواند وجود داشته باشد؛ در سال ۱۹۷۷ مقاله‌ای با عنوان "بیماری یکریختی گراف" منتشر شد که مسئله یکریختی گراف را به‌عنوان مسئله‌ای که هیچ الگوریتم چندجمله‌ای در آینده برای آن وجود نخواهد داشت مطرح کرد[۱۵].

ازآنجاکه حداقل بیش از یک‌صد الگوریتم منتشر شده در این زمینه وجود دارد بررسی پیشینه این مسئله به‌صورت جامع، ممکن نیست ازاین‌رو در ادامه به چند روش از معروف‌ترین روش‌های حل مسئله یکریختی گراف اشاره می‌شود:

- روش برچسب‌گذاری کانونی، متداول‌ترین راه‌حل مسئله یکریختی گراف‌ها و بهینه‌ترین روش شناخته شده برای تشخیص یکریختی میان درخت‌ها است[۱۶]. در این روش یک رویه بر روی گراف ورودی، برچسب‌گذاری مجدد

انجام می‌دهد به‌گونه‌ای که با گراف‌های یکریخت خود خروجی یکسان داشته باشد. هنگامی‌که یک روش برچسب زدن کانونی کارآمد وجود داشته باشد می‌توان از الگوریتم مرتب‌سازی، درهم[۹] سازی یا درخت متعادل برای جداکردن گراف‌های یکریخت در یک مجموعه بزرگ از گراف‌ها استفاده کرد.

- روش قدم زدن کوانتومی[۱۰] که در ده سال گذشته بر روی انواع مختلف گراف مطرح شده و توانسته است با طرح الگوریتم‌های متفاوت در بهترین حالت به پیچیدگی زمانی $O(n^6)$ دست یابد[۱۷]. محققان با آخرین تغییرات ایجاد شده بر روی الگوریتم پیاده‌روی کوانتومی توانسته‌اند به نتایج صد درصد صحیح برای تشخیص یکریختی گراف‌های ساده برسند. تنها نقطه‌ضعف این روش، تشخیص یکریختی گراف‌های منظم نشان داده شده است.
- روش گروه‌بندی گراف‌های خودریخت[۱۱] که به رابطه میان مسئله یکریختی با خودریختی گراف می‌پردازد، از سال ۱۹۹۱ تا حال حاضر تحت مطالعه و بررسی می‌باشد.

با تحقیق و بررسی بر روی تاریخچه راه‌حل‌های موجود درمی‌یابیم که موفقیت‌آمیزترین آن‌ها، شامل تثبیت یک رأس در دو گراف و به همراه آن اعمال تغییرات بر روی مجموعه رأس‌های تثبیت نشده است[۱۸].

سریع‌ترین الگوریتم برای تشخیص یکریختی گراف دارای پیچیدگی زمانی $e^{O(\sqrt{n \log n})}$ است[۱۹]. تنها کلاس‌های خاصی از گراف دارای الگوریتم چندجمله‌ای برای حل مسئله یکریختی گراف می‌باشند، به‌طور نمونه:

- گراف‌های مسطح[۲۰]
- گراف‌های با درجه محدود[۲۱]
- درخت‌ها[۲۲].

۴- الگوریتم پیشنهادی

در روش کدگذاری زمانی می‌توان یکریختی دو گراف را به‌طور قطعی تشخیص داد اگر و فقط اگر:

- برای هر دو گراف غیریکریخت، کدهای متفاوتی تولید شود.
- برای هر دو گراف یکریخت، کدهای یکسانی تولید شود.

در این مقاله با وجود نتایج مثبت حداکثری، هنوز امکان تصمیم‌گیری قطعی در مورد مسئله یکریختی محقق نشده است.

الگوریتم بر اساس مقدار خروج از مرکز یا ecc هر رأس در گراف عمل می‌کند و مطابق آن، رأس‌های گراف را به لایه‌هایی تقسیم می‌کند که در هر لایه، ecc رأس‌ها یکسان است. روند الگوریتم از لایه‌های با ecc بیشتر به سمت لایه‌های با ecc کمتر پیش می‌رود. در هر مرحله ابتدا رأس‌های مجاور داخل هر لایه عملیات انتقال کد پرانتزی را میان خود انجام داده، سپس در درون لایه مجاور خود که دارای ecc یک واحد کمتر می‌باشند عملیات انتقال کد را ادامه می‌دهند. حلقه اصلی الگوریتم عملیات لایه‌بندی گراف می‌باشد که در درون آن عملیات کد گذاری شامل مراحل زیر است:

انتقال کد هر رأس با بالاترین ecc به همسایگان خود که دارای همان مقدار ecc باشند و سپس انتقال کد با بالاترین مقدار ecc به رأس‌های با مقدار ecc که یک واحد کمترند.

- مرتب‌سازی کدهای دریافتی به‌صورت صعودی و بر اساس ترتیب لغوی
- تکرار مراحل فوق تا رئوس با کمترین مقدار ecc

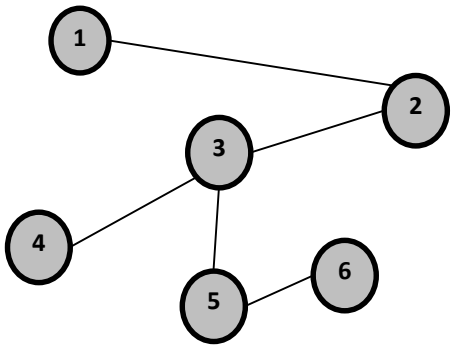
در راه‌حل پیشنهادی که شبه کد آن در شکل ۱ در قالب الگوریتم $GraphIsomorphismCheck()$ نشان داده شده است ابتدا تعداد رأس‌ها و تعداد یال‌ها برای دو گراف ورودی بررسی شده و در صورت یکسان بودن، کد پرانتزی مربوط به هر دو گراف توسط تابع $ParenthesisCode()$ تولید می‌شود و سپس با مقایسه میان رشته‌کدهای دو گراف، یکریختی یا عدم یکریختی آن‌ها مشخص می‌شود.

```
Algorithm GraphIsomorphismCheck (G, G')
input:  G = (V, E), G' = (V', E') // two finite simple connected graphs
output: true   // if G is isomorphic to G'
        false  // else

begin
  if |V| ≠ |V'| or |E| ≠ |E'| then
    return false;
  if ParenthesisCode(G) ≠ ParenthesisCode(G') then
    return false;
  return true;
end of Algorithm.
```

```
function ParenthesisCode (G)
input: G = (V, E) : graph  // a finite simple connected graph where
          // every vertex v has v.ecc:integer, v.oldcode, v.code:string,
          // v.neighbors:set of all adjacent vertices of vertex v
output: GraphCode : string   // parenthesis code of G

begin
1: for all vertex v ∈ V do        // using BFS on all vertices
2:    compute ecc(v) and assign it to v.ecc;
3: minecc ← min_{v∈V}{v.ecc};
4: maxecc ← max_{v∈V}{v.ecc};
5: for all vertices v ∈ V do
6:    v.code ← "()"; v.incode ← " "; //assign empty string to v.incode
7: for i ← maxecc downto minecc do
8:    for all vertices v ∈ V in which v.ecc = i do
9:       v.oldcode ← v.code;
10:   for all vertices v ∈ V in which v.ecc = i do
11:      L ← ∅;  //empty list of string codes
12:      add c to L in which v.oldcode = "(c)"; //c is inner code of v.oldcode
13:      for all u ∈ v.neighbors do
14:         if u.ecc = i then
15:            add u.oldcode to L;
16:      sort(L);// sort all elements of list L as lexicographically
17:      S ← concatenation of all elements of list L;    // S is a string
18:      v.code ← "(" + S + ")"; // concatenate "(" and ")" outer of S
19:   for all vertices v ∈ V in which v.ecc = i − 1 do
20:      v.oldcode ← v.code;
21:   for all vertices v ∈ V in which v.ecc = i − 1 do
22:      L ← ∅;  //empty list of string codes
23:      add c to L in which v.oldcode = "(c)"; //c is inner code of v.oldcode
24:      for all u ∈ v.neighbors do
25:         if u.ecc = i then
26:            add u.oldcode to L;
27:      sort(L);// sort all elements of list L as lexicographically
28:      S ← concatenation of elements of list L;        // S is a string
29:      v.code ← "(" + S + ")"; // concatenate "(" and ")" outer of S
30:L ← ∅  //empty list of sting codes
31:for all vertices v ∈ V do
32:   add v.code to L;
33:sort(L);   // sort all elements of list L as lexicographically
34:GraphCode ← concatenation of all elements of list L;
35:return GraphCode;
end of function;
```

شکل ۱- شبه کد الگوریتم پیشنهادی

در این قسمت، یک نمونه گراف به همراه کد پرانتزی حاصل از اجرای الگوریتم پیشنهادی بر روی آن مطرح می‌شود. گراف شکل ۲ را در نظر بگیرید:



شکل ۲- نمونه گراف ساده همبند با ۶ رأس

گام اول: ecc مربوط به همه رئوس تعیین می‌شود:

$ecc(1) \leftarrow 3,\quad ecc(3) \leftarrow 2,\quad ecc(5) \leftarrow 3,$
$ecc(2) \leftarrow 2,\quad ecc(4) \leftarrow 3,\quad ecc(6) \leftarrow 3.$

سپس کد اولیه هر رأس برابر با "( )" قرار داده می‌شود.

گام دوم: بر روی مجموعه رأس‌ها لایه‌بندی انجام می‌گیرد، رأس‌های با مقدار ecc برابر را در یک لایه $(Layer)$ فرض می‌کنیم.

$Layer_3\ (ecc = 3) = \{1, 4, 5, 6\}$

$Layer_2\ (ecc = 2) = \{2, 3\}$

گام سوم: بر اساس مجاورت رأس‌ها، عملیات انتقال کد انجام می‌شود.

لازم به ذکر است که در زمان اتصال رشته‌کدها برای تولید کد مربوط به هر رأس، عمل مرتب‌سازی لغوی (Lexicographic) انجام می‌شود. کد درون پرانتز باز و بسته یک رأس، کدی است که از همسایگان آن رأس به آن به ارث رسیده است و کد مربوط به همسایگان به‌صورت مرتب لغوی در کنار هم قرار می‌گیرند.

$Layer_3$ to $Layer_3$:

$code(1) \leftarrow$ "( )"
$code(4) \leftarrow$ "( )"
$code(5) \leftarrow$ "(( ))"

$code(6) \leftarrow$ (( ))

$Layer_3$ to $Layer_2$:

$code(2) \leftarrow$ "(( ))"
$code(3) \leftarrow$ "((( ))(( ))( ))"

$Layer_2$ to $Layer_2$:

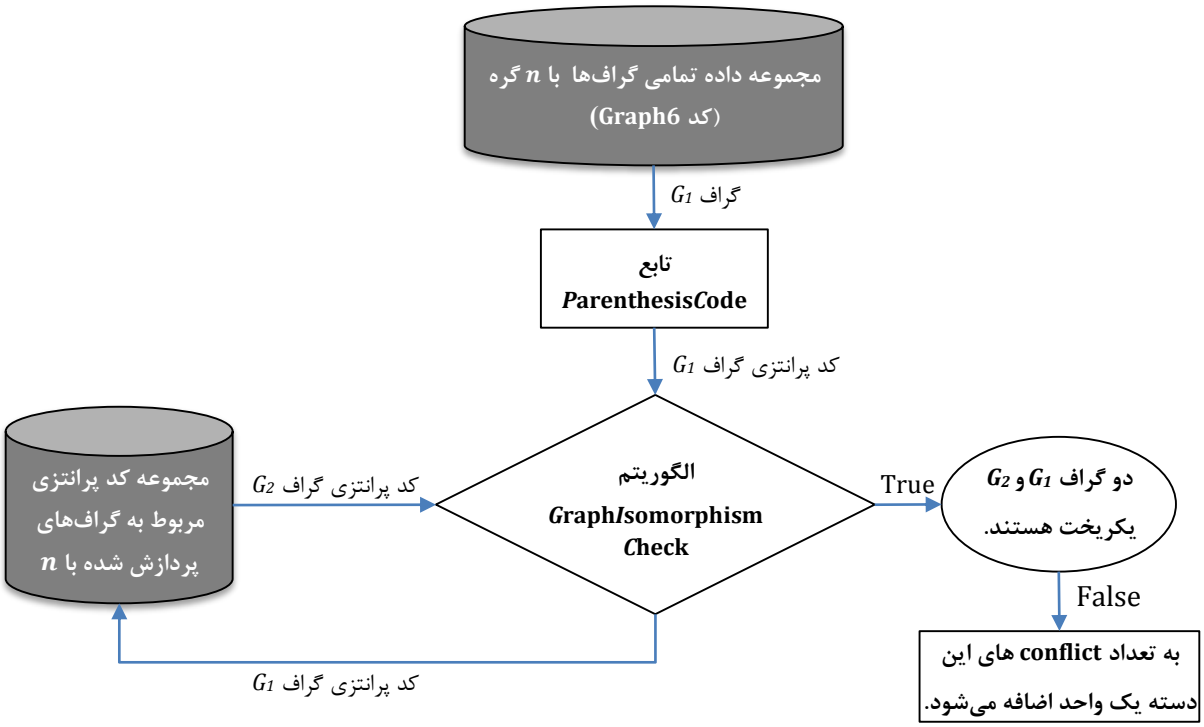
$code(2) \leftarrow$ "(((( ))(( ))( ))( ))"
$code(3) \leftarrow$ "((( ))(( ))(( ))( ))"

گام نهایی: با اتصال برچسب نهایی مربوط به تمامی رأس‌ها، کد مربوط به گراف مشخص می‌شود. کد پرانتزی مربوط به گراف شکل ۲ برابر با رشته زیر است:

"((((( ))(( ))( ))( ))(((( ))(( ))(( ))( ))(( ))(( ))( ))( ))"

پیاده‌سازی الگوریتم: تمامی گراف‌های قرار گرفته در یک دسته، در یک مرحله اجرای الگوریتم در پیاده‌سازی با یکدیگر مقایسه می‌شوند. مراحل پردازش الگوریتم طبق شکل پیاده‌سازی شده است. شکل ۳ یک مرحله از اجرای الگوریتم را نشان می‌دهد.

تحلیل پیچیدگی زمانی الگوریتم پیشنهادی: برای محاسبه مقدار ecc برای همه رئوس می‌توان از هر رأس گراف یک‌بار الگوریتم جستجوی ردیفی(BFS) [۱۲] را اجرا کرد، لذا پیچیدگی زمانی حلقه خط شماره 1 الگوریتم $O(ne)$ می‌باشد که در آن n تعداد رئوس گراف و e تعداد یال‌های گراف است. پیچیدگی خط‌های 3 و 4 نیز هر کدام $O(n)$ خواهند بود. ازآنجاکه مرتب‌سازی لغوی روی کدهای مرتب‌شده انجام می‌شود لذا می‌توان از Bucket Sort استفاده نمود که دارای پیچیدگی زمانی خطی برحسب تعداد کل عناصر است که در کل می‌تواند در $O(e)$ انجام پذیرد. این مرتب‌سازی و الحاق در درون حلقه‌هایی که به تعداد رئوس تکرار می‌شوند قرار دارند و در کل مقدار زمان $O(ne)$ خواهد بود. بنابراین پیچیدگی زمانی الگوریتم درمجموع برابر $O(ne)$ است.



شکل ۳- مدل پیاده‌سازی الگوریتم پیشنهادی

۵- نتایج ارزیابی عملکرد

گراف‌های در نظر گرفته شده برای ورودی الگوریتم، گراف‌های همبند و بدون برچسب می‌باشند که برحسب تعداد رأس دسته‌بندی شده‌اند [۲۳]. مجموعه گراف‌های ورودی بدین شرح است:

- تمامی گراف‌های ساده همبند با تعداد رأس ۳ تا ۹
- تمامی گراف‌های همبند اویلری با تعداد رأس ۵ تا ۱۰
- تمامی گراف‌های همبند وتری با تعداد رأس ۴ تا ۹

همان‌طور که مشاهده می‌شود مجموعه‌های در نظر گرفته شده در هر نوع، شامل تمامی گراف‌های موجود در نوع خود می‌باشد.

حاصل اجرای الگوریتم پیشنهادی بر روی مجموعه اول گراف‌ها، در جدول ۱ آمده است:

جدول ۱- عملکرد الگوریتم بر روی گراف‌های ساده همبند

تعداد رأس	تعداد زوج گراف‌ها	میزان درستی خروجی الگوریتم
۳	۱	۱۰۰٪
۴	۱۵	۱۰۰٪
۵	۲۱۰	۹۹/۵ ٪
۶	۶،۲۱۶	۹۹/۵ ٪
۷	۳۶۳،۳۷۸	۹۹/۷ ٪
۸	۶۱،۷۸۸،۲۸۶	۹۹/۹ ٪
۹	۳۴،۰۸۱،۲۵۲،۶۶۰	۹۹/۹ ٪
۱۰	۳۰،۵۳۹،۱۰۵،۳۶۸	۹۹/۹ ٪

مطابق اطلاعات به‌دست‌آمده در جدول ۱، الگوریتم پیشنهادی بر روی کلیه گراف‌های ساده موجود با تعداد رأس کمتر از ۱۰ دارای صحت تشخیص یکریختی با میانگین بیش از ۹۹/۵ درصد می‌باشد و ازآنجاکه آزمایش بر روی تمامی گراف‌های موجود صورت گرفته است این نتیجه کاملاً قطعی است.

در مراحل بعد گراف‌هایی غیر از گراف‌های ساده برای ورودی الگوریتم در نظر گرفته شد که جدول نتایج حاصل از اجرای الگوریتم بر روی آن‌ها در ادامه آمده است. عملکرد الگوریتم پیشنهادی بر روی گراف‌های اویلر در جدول ۲ نشان داده شده است که مطابق نتایج به‌دست‌آمده، به‌استثناء نتیجه صددرصدی برای گراف‌های با ۵ رأس، درستی عملکرد الگوریتم با افزایش تعداد رأس‌های گراف افزایش یافته است.

جدول ۲- عملکرد الگوریتم بر روی گراف‌های اویلری[۱۳]

تعداد رأس	تعداد زوج گراف‌ها	میزان درستی خروجی الگوریتم
۵	۶	۱۰۰٪
۶	۲۸	۹۶/۴ ٪
۷	۶۶۶	۹۸/۹ ٪
۸	۱۶،۸۳۶	۹۹/۱ ٪
۹	۱،۵۸۶،۸۷۱	۹۹/۴ ٪
۱۰	۴۸۱،۲۹۰،۸۲۵	۹۹/۶ ٪

درستی عملکرد الگوریتم پیشنهادی بر روی بیش از ۷۰ میلیون زوج گراف وتری (جدول ۳) نزدیک به ۱۰۰٪ می‌باشد.

جدول ۳- عملکرد الگوریتم بر روی گراف‌های وتری[۱۴]

تعداد رأس	تعداد زوج گراف‌ها	میزان درستی خروجی الگوریتم
۴	۱۰	۱۰۰٪
۵	۱۰۵	۱۰۰٪
۶	۱،۶۵۳	۹۹/۸ ٪
۷	۳۶،۸۵۶	۹۹/۹ ٪
۸	۱،۳۰۱،۶۹۱	۱۰۰٪
۹	۷۰،۹۳۰،۰۰۵	۱۰۰٪

مقایسه عملکرد الگوریتم: قدرتمندترین الگوریتم موجود در حال حاضر بسته Nauty و Traces است [۲۴]. الگوریتم Nauty [۲۵] و الگوریتم Traces [۲۶] دو الگوریتم مجزا می‌باشند که از سال ۲۰۱۴ به‌صورت یک بسته در کنار هم قرار گرفته‌اند. الگوریتم Nauty و Traces، الگوریتمی جهت محاسبه گروه‌های هم‌ریختی[۱۵] گراف و حل مسئله یکریختی دو گراف با استفاده از روش کدگذاری کانونی می‌باشد. بسته Nauty و Traces تحت سیستم‌عامل لینوکس و با زبان C پیاده‌سازی شده است که به‌صورت موازی پردازش می‌شود.

مقایسه میان دو الگوریتم در دو حالت صورت گرفته است:

الف- صحت عملکرد دو الگوریتم بر روی گراف‌های منظم[۱۶] تصادفی

ب- عملکرد دو الگوریتم ازلحاظ زمان و فضا بر روی گراف کامل

الف- در جداول قبلی عملکرد الگوریتم پیشنهادی در عدم یکریختی میان گراف‌های کوچک بیان شد. ازاین‌رو در این مقایسه درصد صحت تشخیص دو الگوریتم بر روی گراف‌های با مرتبه بیشتر بررسی می‌شود که اطلاعات مربوط به این گراف‌ها از وب‌گاه "The House of Graphs" گرفته شده است [۲۷]. جدول ۴ نتایج حاصل از این مقایسه را نشان می‌دهد.

جدول ۴- عملکرد الگوریتم پیشنهادی و بسته Nauty و Traces بر روی گراف‌های ساده با اندازه متوسط

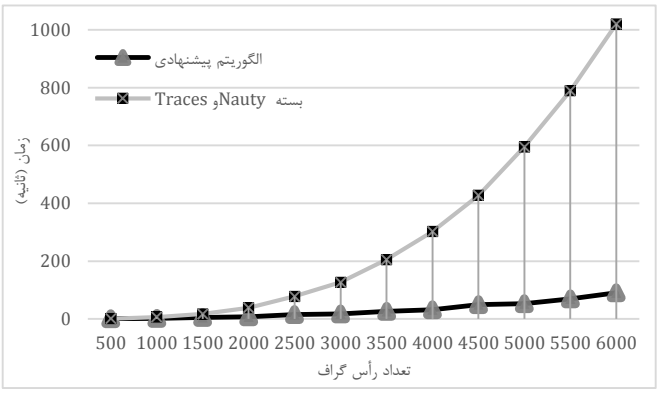
تعداد رأس	تعداد گراف‌ها	میزان درستی خروجی الگوریتم پیشنهادی	میزان درستی خروجی بسته Nauty و Traces
۲۰	۵۵۵	۹۹/۹۲ ٪	۱۰۰٪
۳۰	۵۷۳	۹۹/۳۲ ٪	۱۰۰٪
۴۰	۳۵۵	۹۹/۹۶ ٪	۱۰۰٪
۵۰	۱۵	۱۰۰٪	۱۰۰٪
۶۰	۲۱	۱۰۰٪	۱۰۰٪

با توجه به نتایج بالا الگوریتم پیشنهادی دارای بازدهی خوبی در مجموعه گراف‌های با مرتبه نسبتاً بالا می‌باشد همچنین می‌توان به قدرت بالای بسته Nauty و Traces در تشخیص عدم یکریختی پی برد.

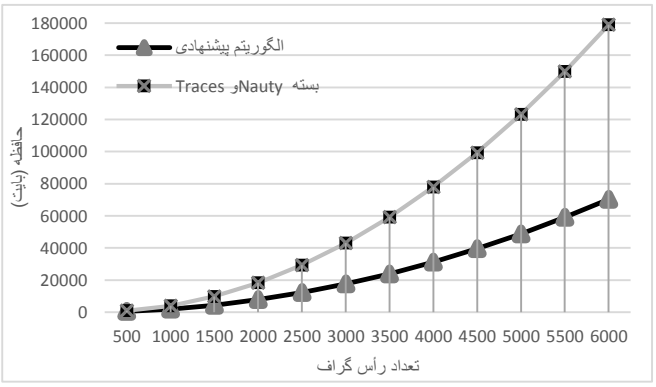
ب- در این دو مقایسه دو الگوریتم از نظر زمان اجرا و فضای مصرفی مقایسه می‌شوند. جدول ۳ درصد صحت تشخیص دو الگوریتم را در عدم یکریختی میان مجموعه‌ای از گراف‌های منظم نشان می‌دهد.

نمودار شکل ۴ مربوط به زمان اجرای الگوریتم پیشنهادی و بسته Nauty و Traces و نمودار شکل ۵ مربوط به فضای اشغال شده توسط آن‌ها می‌باشد و داده‌ها متوسط به‌دست‌آمده پس از سه بار اجرا است.

داده‌های مربوط به نمودارها از اجرای هر دو الگوریتم بر روی یک سیستم با پردازنده ۲/۵ گیگاهرتز ۵ هسته‌ای و حافظه ۸ گیگابایت با ورودی یکسان به‌دست‌آمده است. گراف مورد آزمایش برای ورودی الگوریتم‌ها گراف کامل[۱۷] انتخاب شده است.



شکل ۴- نمودار مقایسه زمان اجرای الگوریتم پیشنهادی و بسته Nauty و Traces



شکل ۵- نمودار مقایسه میزان حافظه مورداستفاده الگوریتم پیشنهادی و بسته Nauty و Traces

ارزیابی بر روی گراف کامل انجام شده است زیرا گراف کامل بیشترین تعداد یال را با تعداد رأس n دارد و بررسی یکریختی میان دو گراف کامل بیشترین زمان و فضا را صرف می‌کند.

دو گراف کامل G_1 و G_2 توسط توابع ایجاد گراف در محیط‌های پیاده‌سازی الگوریتم‌ها ایجاد شدند:

- G_1 گراف کامل با n رأس
- G_2 گراف کامل با n رأس

چون در حالت اولیه دو گراف کامل با n رأس باهم یکریخت هستند صحت تشخیص یکریختی میان دو گراف در هر دو الگوریتم در تمام موارد و در هر سه اجرا برابر ٪۱۰۰ بود.

زمان اجرای اندازه‌گیری شده در نمودار شکل ۴ شامل زمان تولید کد کانونی برای یک گراف به اضافه زمان مقایسه کد تولید شده گراف اول با کد موجود گراف دوم و تشخیص یکریختی میان دو گراف است به‌بیان‌دیگر در الگوریتم پیشنهادی این زمان برابر

$$T_{Run} = T_{ParenthesisCode}(G_1) + T_{GraphIsomorphismCheck}(G_1, G_2)$$

و در بسته Nauty و Traces برابر

$$T_{Run} = T_{canonical\ code}(G_1) + T_{GraphIsomorphismCheck}(G_1, G_2)$$

می‌باشد. با توجه به شکل ۴، نمودار زمان اجرای بسته Nauty و Traces با افزایش مرتبه گراف، زمان بیشتری را برای ایجاد کد کانونی و تشخیص یکریختی صرف کرده است و میان نمودار زمان اجرای دو الگوریتم با افزایش تعداد رأس‌ها فاصله بیشتری به وجود آمده است. الگوریتم پیشنهادی در این بازه که مورد آزمایش قرار گرفته است دارای سرعت بالاتری می‌باشد. لازم به ذکر است که آخرین تغییرات در پیاده‌سازی بسته Nauty و Traces در فایل‌های متنی از طریق سایت مربوطه در دسترس کاربران قرار دارد [۲۴] و مقادیر موجود در شکل ۴ از طریق ابزار معرفی شده در همین سایت با نام $dreadnaut.exe$ به‌دست‌آمده‌است.

برای اندازه‌گیری میزان حافظه هر دو الگوریتم، خروجی تولید شده برای یک گراف ورودی در یک فایل متنی[۱۸] ذخیره شده و اندازه فایل‌های ایجاد شده در هر دو الگوریتم با یکدیگر مقایسه شده‌اند؛ خروجی الگوریتم پیشنهادی، رشته‌کد پرانتزی مربوط به یک گراف و خروجی بسته Nauty و Traces، داده‌های مربوط به ماتریس مجاورت یک گراف می‌باشد. با توجه به دو نمودار مربوط به الگوریتم‌ها در شکل ۵، الگوریتم پیشنهادی دارای میزان حافظه مصرفی کمتری نسبت به بسته Nauty و Traces می‌باشد.

در مجموع می‌توان گفت که بسته Nauty و Traces از نظر صحت عملکرد نسبت به الگوریتم پیشنهادی عملکرد بهتری دارد و الگوریتم پیشنهادی ازلحاظ سرعت اجرا و میزان حافظه اشغالی در بازه‌های آزمایش شده در شکل ۴ و شکل ۵ نتایج بهتری را نشان می‌دهد.

۶- نتیجه‌گیری

در این مقاله با ارائه الگوریتمی ابتکاری و درعین‌حال ساده که برای اجرا بر روی سیستم به حداقل حافظه و پردازش نیاز دارد، یکریختی میان دو گراف متناهی قابل تشخیص است. الگوریتم پیشنهادی، تکنیکی جدید در کدگذاری کانونی یک گراف ارائه می‌دهد که با استفاده از ویژگی فاصله از مرکز رأس‌ها آن‌ها را لایه‌بندی می‌کند و در انتها کد پرانتزی مربوط به گراف را تشکیل می‌دهد. پیاده‌سازی الگوریتم بر اساس جداول نتیجه با استفاده از کد ایجاد شده به مقایسه گراف‌ها و تشخیص یکریختی آن‌ها می‌پردازد. بر اساس اطلاعات جدول ۱ که حاصل اجرای الگوریتم بر روی حداقل ۶۴ میلیارد زوج گراف ساده همبند می‌باشد، میزان عملکرد صحیح الگوریتم بیشتر از ٪۹۹/۵ می‌باشد. این مسئله در نگاشت الگوریتم‌های موازی از یک توپولوژی در شبکه‌ی بین پردازنده‌ای به توپولوژی دیگر ارتباط مستقیم دارد.

نقطه‌ضعف این الگوریتم در تشخیص یکریختی میان دو گراف منظم خود مرکز با تعداد رأس برابر و تعداد یال برابر می‌باشد که با توجه به نحوه تشکیل کد پرانتزی، این الگوریتم برای تمامی گراف‌های منظم خود مرکز غیر یک‌ریخت با درجه رأس یکسان، رشته کد یکسان تولید کرده و همگی را یک‌ریخت تشخیص می‌دهد. بنابراین برای توسعه الگوریتم می‌توان بر روی مواردی که ecc تمام رأس‌های گراف یکسان است به‌بیان‌دیگر روی گراف‌های خود مرکز، تغییراتی در الگوریتم ایجاد کرد.

اجرای این الگوریتم بر روی گراف‌های با شرایط خاص نیز نتایج بسیار خوبی را تولید کرده است که در بخش ۵ نشان داده شد. امتیاز الگوریتم پیشنهاد شده در این مقاله نسبت به الگوریتم‌های مطرح شده در سال‌های اخیر برای مسئله یکریختی عبارت است از:

- در الگوریتم پیشنهادی کلیه گراف‌های شناخته شده به‌عنوان گراف ورودی مورد بررسی قرار گرفته درحالی‌که در بیشتر الگوریتم‌ها عملکرد آن‌ها با ورودی گراف‌های تصادفی سنجیده شده است.
- پیچیدگی زمانی اجرای الگوریتم پیشنهادی به‌صورت قطعی مشخص می‌باشد درصورتی‌که پیچیدگی زمانی الگوریتم‌های دیگر طبق اجراهای متناوب و با واحد زمانی سنجیده می‌شود.

الگوریتم پیشنهادی در بیش از ٪۹۹ حالات مختلف، در گراف‌های تا ۱۰ گره کاملاً قابل‌اتکا بوده و دارای مرتبه زمانی $O(ne)$ می‌باشد.

۷- مراجع

[1] S. Wasserman, and K. Faust, "Social Network Analysis: Methods and Applications", *9th ed., United Kingdom: Cambridge University Press*, 1994.

[2] Th. Coffman, S. Greenblatt, and Sh. Marcus, "Graph-based technologies for intelligence", *Communications of the ACM*, Vol. 47, No. 3, pp. 45–47, 2004.

[3] V. Lacroix, C. G. Fernandes, and M. F. Sagot, "Motif search in graphs: application to metabolic networks", *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, Vol. 3, No. 4, pp. 360–368, 2006.

[4] T. Aittokallio, and B. Schwikowski, "Graph-based methods for analysing networks in cell biology", *Briefings in Bioinformatics*, Vol. 7, No. 3, pp. 243–255, 2006.

[5] M. Terzer, N. D. Maynard, M. W. Covert, and J. Stelling, "Genome-scale metabolic networks", *WIREs: Systems Biologhy and Medicine*, Vol. 1, No. 3, pp. 285–297, 2009.

[6] L. P. Cordella, P. Foggia, and C. Sansone, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, No. 10, pp. 1367–1372, 2004.

[7] M. A. Abdulrahim, and M. Misra, "A Graph Isomorphism Algorithm for Object Recognition", *Pattern Analysis and Applications*, Vol. 1, No. 3, pp. 189–201, 1998.

[8] A. Aparo, V. Bonnici, G. Micale, A. Ferro, D. Shasha, A. Pulvirenti, and R. Giugno, "Fast Subgraph Matching Strategies Based on Pattern–Only Heuristics", *Interdisciplinary Sciences: Computational Life Sciences*, Vol. 11, No. 1, pp. 21–32, 2019.

[9] R. M. Karp, "*Reducibility Among Combinatorial Problems*", R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). *Complexity of Computer Computations,* New York: Plenum. pp. 85–103, 1972.

[10] S. D. Stoichev, "*New Exact and Heuristic Algorithms for Graph Automorphism Group and Graph Isomorphism*", Journal of Experimental Algorithmics (JEA), Vol. 24, No. 1, pp. 1-15, 2019.

[11] B. D. McKay, and A. Piperno, "*Practical graph isomorphism, II*", Journal of Symbolic Computation, Vol. 60, pp. 94-112, 2014.

[12] V. M. V. Rachna Somkunwar, "*A Comparative Study of Graph Isomorphism Applications*", International Journal of Computer Applications, Vol. 162, No. 7, pp. 34-37, 2017.

[13] G. Chartrand, L. Lesniak, and P. Zhang, "*Graphs & Digraphs",* (6th ed.). Chapman & Hall/CRC, 2015.

[14] A. Hou, Q. Zhong, Y. Chen, and Zh. Hao, "A Practical Graph Isomorphism Algorithm with Vertex Canonical Labeling", *Journal of Computers (JCP)*, Vol. 9, No. 10, pp. 2467–2474, 2014.

[15] R. C. Read, D. G. Corneil, "*The graph isomorphism disease*", Graph Theory, Vol. 1, No. 4, pp. 339-363, 1977.

[16] P. D. G. Valiente, "*Algorithms on Trees and Graphs*", *Springer*, 2002.

[17] K. Liu, Y. Zhang, K. Lu, X. Wang, X. Wang, and G. Tian, "*MapEff: An Effective Graph Isomorphism Agorithm Based on the Discrete-Time Quantum Walk*", Entropy, Vol. 21, No. 6, pp. 569-584, 2019.

[18] T. P. Zivković, "*On graph isomorphism and graph automorphism*", Journal of Mathematical Chemistry, Vol. 8, No. 1, pp. 19–37, 1991.

[19] L. Babai, "*Graph isomorphism in quasipolynomial time*", *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing* (STOC '16). ACM, New York, NY, USA, 684-697, 2016.

[20] J. E. Hopcroft, and J. Wong, "*Linear time algorithm for isomorphism of planar graphs*", Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 172–184, 1974.

[21] E. M. Luks, "Isomorphism of graphs of bounded valence can be tested in polynomial time", *Journal of Computer and System Sciences*, Vol. 25, pp. 42–65, 1982.

[22] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "*The Design and Analysis of Computer Algorithms*", Addison-Wesley, 1974.

[23] ANU College of Engineering & Computer Science. *"collections of non-isomorphism graphs"*, Available: http://users.cecs.anu.edu.au/~bdm/data/graphs.html.

[24] Graph canonical labeling and automorphism group computation. "Nauty and Traces", Available: http://pallini.di.uniroma1.it/index.html.

[25] B. D. McKay, "Practical Graph Isomorphism", *Congressus Numerantium*, Vol. 30, pp. 45-87, 1981.

[26] A. Piperno, "Search space contraction in canonical labeling of graphs", preprint 2008 (v2: 2011). Available: http://arxiv.org/abs/0804.4881.

[27] The House of Graphs. "Database of interesting graphs", Available: https://hog.grinvin.org/Init.action.

علی نوراله مدرک کارشناسی ارشد و دکتری خود را به ترتیب در سال‌های ۱۳۷۷ و ۱۳۸۷ در رشته مهندسی کامپیوتر (نرم‌افزار) از دانشگاه صنعتی شریف و دانشگاه صنعتی امیرکبیر (پلی‌تکنیک تهران) دریافت نموده است. ایشان در کلیه مقاطع تحصیلی (کارشناسی، کارشناسی ارشد و دکتری) دارای رتبه اول و ممتاز بوده است. زمینه تحقیقاتی ایشان الگوریتم‌های هندسه محاسباتی و حرکت روبات‌ها می‌باشد. ایشان در حال حاضر عضو هیئت‌علمی دانشگاه تربیت دبیر شهید رجایی و دارای رتبه استادیاری می‌باشد و دروسی نظیر اصول طراحی کامپایلر، نظریه زبان‌ها و ماشین‌ها، طراحی و پیاده‌سازی زبان‌های برنامه‌سازی، طراحی و تحلیل الگوریتم‌ها و ساختمان داده‌ها را در دوره کارشناسی و دروسی نظیر الگوریتم‌های هندسه محاسباتی، الگوریتم‌های موازی پیشرفته، الگوریتم‌های گراف و اصول رمزنگاری را در دوره کارشناسی ارشد تدریس می‌نماید.

آدرس پست الکترونیکی ایشان عبارت است از:

nourollah@aut.ac.ir

سمیه چک در حال حاضر دانشجوی کارشناسی ارشد مهندسی کامپیوتر (نرم‌افزار) می‌باشد که زمینه تحقیقاتی وی الگوریتم‌های تشخیص یکریختی در درخت‌ها و گراف‌ها می‌باشد.

آدرس پست الکترونیکی ایشان عبارت است از:

somayeh_check@sru.ac.ir

[1] Heuristic Algorithms
[2] Canonical Labeling
[3] Simple
[4] Connected
[5] Unlabeled
[6] Non-Isomorphism Graphs
[7] Eccentricity
[8] Self-Centred
[9] Hash
[10] Quantum Walk
[11] Automorphism
[12] Breadth First Search
[13] Eulerian Graph
[14] Chordal Graph
[15] Automorphism
[16] Regular Graphs
[17] Complete Graph
[18] text file

The CSI Journal on Computing Science and Information Technology

The CSI Journal on
Computing Science and
Information Technology
Vol. 17, No.2, 2020

A Heuristic Algorithm for Graphs Isomorphism Problem Based on Graph Eccentricity

Ali Nourollah, Somayeh Check

Faculty of Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran

Abstract

Graph Isomorphism Problem (GIP) is an open problem due to computational complexity that only its belonging to the NP class is known, but it does not belong to P or NP-Complete. There is no deterministic polynomial time algorithm proposed yet, therefore, the field is open for research and ideas. The heuristic and meta heuristic approaches have been the only way to solve it. This paper discusses ways to determine isomorphism between two finite graphs and proposes a novel heuristic method. The proposed algorithm converts the input graph into a parenthesized string, and then compares two generated strings instead of comparing two graphs, in order to determine isomorphic or non-isomorphic between two given graphs. Time complexity of the proposed algorithm is O(n.e) where n is the number of vertices of the graph and e is the number of edges of the graph and it is in the category of canonical labelling algorithms for simple, connected and unlabeled graphs. After the implementation of the algorithm and checking its results, it was found that with correct performance of more than 99%, the lack of isomorphic between two non-isomorphic graphs correctly diagnosed.

Keywords: Graph Isomorphism, Polynomial Algorithm, Canonical Labelling, connected simple graph, heuristic algorithm