نوزدهمین کنفرانس ملی سالانه انجمن کامپیوتر ایران
دانشکده مهندسی کامپیوتر، دانشگاه شهید بهشتی، تهران، ۱۵-۱۳ اسفندماه ۱۳۹۲

ارائه یک الگوریتم حریصانه جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار

احمد تاجدینی[۱]، جابر کریم‌پور[۲]، مهدی بیات[۳]

[۱] گروه علوم کامپیوتر، دانشگاه تبریز، تبریز
mehrta.misc@live.com

[۲] استادیار، گروه علوم کامپیوتر، دانشگاه تبریز، تبریز
karimpour@tabrizu.ac.ir

[۳] گروه علوم کامپیوتر، دانشگاه تبریز، تبریز
mehdi.byt@gmail.com

چکیده

در این مقاله به ارائه یک الگوریتم جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار می‌پردازیم. این الگوریتم بدون استفاده از نقاط کمکی، یک چندضلعی را به صورت یکنواخت تجزیه می‌کند. هدف از ارائه این الگوریتم، دستیابی به تجزیه‌ای تقریبا کمینه در یک زمان قابل قبول است. تجزیه کمینه یکنواخت یک چندضلعی حفره‌دار، در حالتی که استفاده از نقاط کمکی مجاز نیست، یک مسئله NP-سخت است. استفاده از الگوریتم‌هایی که منجر به تجزیه تقریبا کمینه می‌شوند، یکی از راهکارهای عملی است.

در طراحی این الگوریتم، دو مقوله کمینگی تجزیه و زمان اجرا مورد توجه قرار گرفته است. این الگوریتم به صورت حریصانه تلاش می‌کند عمل تجزیه را به صورت کمینه انجام دهد. با وجود اینکه تضمینی در مورد بدست آمدن جواب کمینه وجود ندارد، اما نتایج پیاده‌سازی این الگوریتم و مقایسه عملی آن با برخی از الگوریتم‌های موجود، موثر بودن این راهکار را نشان می‌دهد. بخشی از زمان اجرای این الگوریتم با استفاده از پارامتری به نام «حداکثر عمق جستجو» کنترل می‌شود. با تنظیم مقدار این پارامتر و با توجه به کاربرد مسئله، می‌توان بین زمان اجرا و کمینه بودن تجزیه یکی را ترجیح داد و یا اینکه تعادلی بین آنها بوجود آورد.

کلمات کلیدی

تجزیه یکنواخت چندضلعی، چندضلعی یکنواخت، چندضلعی ساده، چندضلعی حفره‌دار، الگوریتم حریصانه

۱- مقدمه

تجزیه چندضلعی فرآیندی است که طی آن یک چندضلعی به مجموعه‌ای از چندضلعی‌های کوچکتر افراز می‌شود. در اکثر موارد لازم است عمل تجزیه به گونه‌ای انجام شود که چندضلعی‌های حاصل از آن، محدب و یا نسبت به یک خط خاص یکنواخت[۱] باشند. در برخی مواقع لازم و یا مطلوب است که تجزیه به صورت کمینه انجام شود.

کمینه بودن تجزیه از دو دیدگاه بررسی می‌شود. در دیدگاه اول، منظور از کمینه بودن تجزیه، کمینه بودن تعداد چندضلعی‌های حاصل از تجزیه است. در دیدگاه دوم، کمینه بودن تجزیه به معنای کمینه بودن مجموع طول قطرهای اضافه شده به چندضلعی است. به این نوع تجزیه، تجزیه با کمترین میزان مصرف جوهر[۲] گفته می‌شود. در این مقاله، منظور از کمینه بودن، کمینه بودن تعداد چندضلعی‌های حاصل از تجزیه است.

تجزیه چندضلعی‌ها در حوزه‌های مختلفی مانند گرافیک برداری، تشخیص الگو، تشخیص متن، محاسبه جمع‌های مینکوفسکی و طرح‌ریزی حرکت ربات کاربرد دارد. از آنجا که اعمال کردن اکثر الگوریتم‌ها روی چندضلعی‌های محدب یا یکنواخت، ساده‌تر و کم هزینه‌تر از اعمال کردن آنها روی چندضلعی‌های معمولی است، تمایل

زیادی برای یافتن الگوریتم‌های کارا جهت تجزیه چندضلعی‌ها به صورت یکنواخت و یا محدب وجود دارد. تا به حال الگوریتم‌های مختلفی برای حل این مسئله ارائه شده است. در این مقاله به ارائه یک الگوریتم حریصانه[۳] جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته می‌شود. این الگوریتم از نقاط کمکی[۴] برای تجزیه چندضلعی استفاده نمی‌کند.

۲- کارهای انجام شده

تجزیه چندضلعی‌ها یکی از مباحث مهم و با سابقه طولانی در حوزه هندسه محاسباتی است. تا به حال الگوریتم‌های متعددی برای تجزیه یک چندضلعی به چندضلعی‌های ساده، یکنواخت، محدب، ستاره‌ای و ذوزنقه‌ای ارائه شده است. پیچیدگی محاسباتی الگوریتم‌های تجزیه چندضلعی، معمولا تحت تاثیر سه عامل قرار دارد. عامل اول تعداد راس‌های چندضلعی است. عامل دوم، تعداد راس‌هایی است که مقدار زاویه داخلی آنها بیشتر از ۱۸۰ درجه می‌باشد. این راس‌ها، راس بازتابی نامیده می‌شوند. عامل سوم تعداد حفره‌های چندضلعی است. در این مقاله، تعداد راس‌های چندضلعی با n، تعداد راس‌های بازتابی با N و تعداد حفره‌های چندضلعی با h نمایش داده می‌شود.

برخی از الگوریتم‌های تجزیه، از نقاط کمکی جهت تسهیل عملیات تجزیه استفاده می‌کنند. این نقاط راس‌هایی می‌باشند که طی فرآیند تجزیه، به مجموعه راس‌های چندضلعی اضافه می‌شوند. با توجه به اینکه این نقاط تعداد راس‌های چندضلعی را افزایش می‌دهند، مطلوب است از این نقاط استفاده نشود یا اینکه میزان استفاده از آنها محدود شود.

برای تجزیه یک چندضلعی ساده، *لی* و *پریپاراتا* یک الگوریتم با زمان اجرای $O(n \log n)$ ارائه داده‌اند [۱]. این الگوریتم از نقاط کمکی استفاده نمی‌کند. تا کنون سه الگوریتم جهت تجزیه کمینه چندضلعی‌ها ارائه شده است. برای حالتی که استفاده از نقاط کمکی مجاز نیست، *کیل* یک الگوریتم با زمان اجرای $O(Nn^4)$ و *لیو* و *نتافوس* یک الگوریتم با زمان اجرای $O(nN^3 + N^2 n \log n + N^5)$ جهت تجزیه کمینه یک چندضلعی ساده ارائه داده‌اند [۲]. چنانچه استفاده از نقاط کمکی مجاز باشد، الگوریتم *لیو* و *نتافوس* (با اعمال تغییراتی) به یک الگوریتم با پیچیدگی محاسباتی $O(nN^3 \log n + N^5)$ تبدیل می‌شود. *کیل* همچنین یک الگوریتم از مرتبه $O(Nn^4)$ برای حل مسئله تجزیه کمینه یک چندضلعی ساده به چندضلعی‌های یکنواخت با کمترین میزان مصرف جوهر، ارائه کرده است [۳].

مسئله تجزیه یک چندضلعی ساده، حالت خاصی از مسئله تجزیه چندضلعی‌های حفره‌دار است (حالتی که $h=0$ است). *وی* یک الگوریتم با زمان اجرای $O(K(n \log n + h\log^3 h))$ برای تجزیه کمینه یک چندضلعی حفره‌دار با استفاده از نقاط کمکی، ارائه کرده است [۴]. در عبارت اخیر، K تعداد اضلاع گراف آشکاری[۵] چندضلعی را نشان می‌دهد. *کیل* نشان داد مسئله تجزیه کمینه یک چندضلعی حفره‌دار به چندضلعی‌های یکنواخت، هنگامی که استفاده از نقاط کمکی مجاز نیست، یک مسئله NP-سخت است [۳].

۳- تعاریف اولیه

در این بخش به مرور برخی از تعاریف مقدماتی مورد نیاز در رابطه با چندضلعی‌ها می‌پردازیم. به هر خط شکسته بسته چندضلعی گفته می‌شود. چندضلعی که اضلاع آن یکدیگر را قطع نکنند و دارای حفره نباشد، چندضلعی ساده نامیده می‌شود. یک چندضلعی ساده نسبت به خط L یکنواخت نامیده می‌شود اگر و تنها اگر هر خط عمود بر L سطح چندضلعی را حداکثر یک بار قطع کند. یک چندضلعی که نسبت به محور Y یکنواخت باشد، چندضلعی یکنواخت قائم[۶] نامیده می‌شود. یکی از ویژگی‌های چندضلعی‌های یکنواخت قائم این است که اگر راس‌های چندضلعی از بالاترین راس به سمت پایین‌ترین راس پیمایش شود آنگاه جهت پیمایش همواره به سمت پایین و یا افقی است و هیچگاه به سمت بالا نیست.

راس‌های یک چندضلعی با توجه به مقدار زاویه داخلی و همچنین موقعیت نسبی آنها، به پنج دسته تقسیم می‌شوند. راسی که یکی از همسایه‌های آن در بالا و همسایه دیگر آن در پایین آن واقع شده باشد، راس معمولی نامیده می‌شود. راسی که معمولی نباشد، راس چرخشی[۷] نامیده می‌شود. راس‌های چرخشی به چهار دسته تقسیم می‌شوند:

- راس شروع: راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن کمتر از ۱۸۰ درجه است.
- راس پایان: راسی که از دو راس مجاور خود پایین‌تر و زاویه داخلی آن کمتر از ۱۸۰ درجه است.
- راس ادغام: راسی که از دو راس مجاور خود پایین‌تر و زاویه داخلی آن بیشتر از ۱۸۰ درجه است.
- راس تفکیک: راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن بیشتر از ۱۸۰ درجه است.

چندضلعی‌هایی که دارای راس ادغام یا تفکیک می‌باشند، یکنواخت قائم نیستند [۱]. به منظور تجزیه یک چندضلعی به مجموعه‌ای از چندضلعی‌های یکنواخت قائم، هر راس ادغام توسط یک قطر به راسی که در پایین آن قرار دارد متصل می‌شود. همچنین هر راس تفکیک توسط یک قطر به راسی که در بالای آن قرار دارد متصل می‌شود. یک تجزیه کمینه زمانی حاصل می‌شود که از کمترین تعداد قطرها استفاده شود. برای اینکه تعداد قطرهای مورد استفاده کمینه

شود، باید تا جایی که امکان دارد زوج راس‌های ادغام-تفکیک بیشتری را توسط یک قطر به یکدیگر متصل کرد. در این حالت، به ازای هر قطر، یک راس تفکیک و یک راس ادغام به صورت همزمان از بین می‌رود.

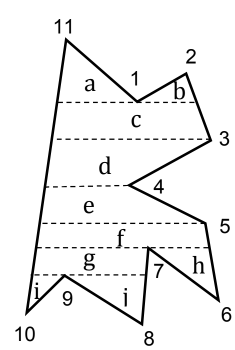
۴- الگوریتم پیشنهادی

در این بخش به ارائه یک الگوریتم جهت تجزیه یکنواخت یک چندضلعی حفره‌دار، بدون استفاده از نقاط کمکی، می‌پردازیم. این الگوریتم الزاما تجزیه را به صورت کمینه انجام نمی‌دهد اما دو هدف اصلی دارد. هدف اول این است که تا جایی که امکان دارد فرآیند تجزیه به سرعت انجام شود. هدف دوم کاهش تعداد چندضلعی‌های حاصل از فرآیند تجزیه است. الگوریتم پیشنهادی ما، با استفاده از تجزیه ذوزنقه‌ای چندضلعی، تلاش می‌کند تا جایی که امکان دارد راس‌های ادغام و تفکیک بیشتری را به یکدیگر متصل کند.

۴-۱- حذف راس‌های ادغام و تفکیک

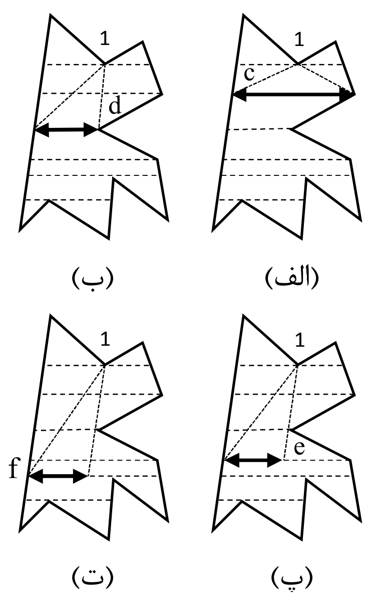
در ادامه به ارائه روشی می‌پردازیم که با استفاده از تجزیه ذوزنقه‌ای یک چندضلعی، سعی می‌کند یک راس ادغام و یک راس تفکیک را به یکدیگر متصل کند. برای تشریح عملکرد این روش، مراحل اجرای آن را روی یک چندضلعی ساده دنبال می‌کنیم. در شکل (۱) یک چندضلعی که به صورت ذوزنقه‌ای تجزیه شده است رسم شده است. این چندضلعی به ۱۰ ذوزنقه تجزیه شده است (مثلثها نیز ذوزنقه‌هایی فرض شده‌اند که طول یکی از قاعده‌های آنها صفر است). ذوزنقه‌هایی که زیر یک راس ادغام قرار می‌گیرند دارای دو همسایه بالایی می‌باشند. به همین صورت، ذوزنقه‌هایی که بالای یک راس تفکیک قرار گرفته‌اند دارای دو همسایه پایینی می‌باشند.

از تجزیه ذوزنقه‌ای می‌توان برای اتصال زوج راس‌های ادغام-تفکیک کمک گرفت. فرض کنید به دنبال راس تفکیکی هستیم که بتوان آن را با استفاده از یک قطر به راس ادغام شماره ۱ متصل کرد. برای انجام این کار می‌توان با شروع از ذوزنقه زیر این راس، یعنی ذوزنقه c، مجموعه ذوزنقه‌ها را به سمت پایین پیمایش کرد.



شکل (۱): یک چندضلعی که به صورت ذوزنقه‌ای تجزیه شده است

در هنگام پیمایش ذوزنقه‌ها، هرگاه با راس تفکیکی (مجاور با قاعده پایینی یک ذوزنقه) مواجه شویم که بتوان آن را به راس شماره ۱ متصل کرد، این کار را انجام داده و پیمایش را متوقف می‌کنیم. هنگام پیمایش ذوزنقه‌ها به سمت پایین، باید به این نکته توجه داشت که ممکن است تنها بخشی از قاعده پایینی ذوزنقه‌ای که ملاقات می‌کنیم، از راس شماره ۱ آشکار[8] باشد. در هنگام ملاقات هر ذوزنقه، بخشی از قاعده پایینی آن که از راس ادغام آشکار است محاسبه می‌شود. شکل (۲) ناحیه آشکار از راس شماره ۱روی قاعده پایینی ذوزنقه‌های c، d، e و f را نشان می‌دهد.



شکل (۲): بازه آشکار از راس شماره ۱ روی قاعده پایینی ذوزنقه‌های c، d، e و f

چنانچه هنگام پیمایش ذوزنقه‌ها به سمت پایین، با ذوزنقه‌ای مواجه شویم که زیر آن یک راس تفکیک قرار دارد، مانند ذوزنقه‌های f و g، ابتدا این موضوع بررسی می‌شود که آیا راس تفکیک در بازه قابل دید از راس ادغام قرار گرفته است و یا خیر. اگر این راس در این بازه قرار گرفته باشد، با استفاده از یک قطر راس ادغام و تفکیک به یکدیگر متصل می‌شوند و تمامی ذوزنقه‌های بین دو راس به دو قسمت تقسیم می‌شوند. اگر راس تفکیک در این بازه قرار نداشته باشد، جستجو به سمت پایین ادامه پیدا می‌کند. باید به این موضوع توجه داشت که در این شرایط ذوزنقه‌ای که در حال ملاقات آن هستیم دارای دو همسایه پایینی سمت چپ و سمت راست است. در این حالت ذوزنقه بعدی که باید ملاقات شود، ذوزنقه‌ای است که بازه قابل دید ذوزنقه فعلی بالای آن قرار گرفته است. بازه قابل دید برای هر ذوزنقه در $O(1)$ و به صورت افزایشی قابل محاسبه است، به این معنا که با داشتن بازه قابل

دید برای یک ذوزنقه، می‌توان بازه قابل دید برای ذوزنقه بعدی را محاسبه کرد. هرگاه طول بازه قابل دید روی ذوزنقه‌ای که در حال ملاقات آن هستیم به صفر برسد، جستجو متوقف می‌شود.

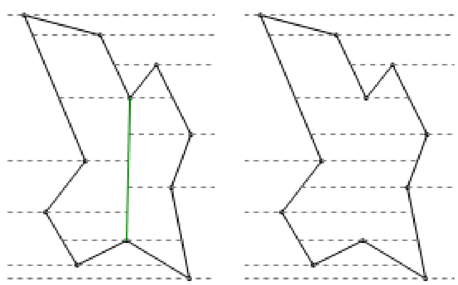
دلیل اصلی استفاده از این روش، قابلیت محدود کردن عمق جستجو است. برای کنترل زمان اجرای الگوریتم، می‌توان حداکثر مقداری را برای تعداد ذوزنقه‌هایی که پیمایش می‌شوند در نظر گرفت. این مقدار را با K نمایش می‌دهیم و آن را حداکثر عمق جستجو می‌نامیم. اگر بعد از پیمایش K ذوزنقه راس تفکیکی یافت نشود، عملیات جستجو خاتمه پیدا می‌کند و راس ادغام در زمان O(1) به یک راس معمولی متصل می‌شود. زمان اجرای کل این عملیات از مرتبه O(K) است. در حالت کلی وقتی مقدار K یک عدد ثابت در نظر گرفته می‌شود، زمان جستجو برای یک راس تفکیک از مرتبه O(1) است.

در الگوریتم (۱) رویه HandleMergeVertex یک راس ادغام را به عنوان پارامتر دریافت می‌کند و سعی می‌کند آن را به یک راس تفکیک متصل کند. اگر راس تفکیکی پیدا نشود (با توجه به مقدار K) راس ادغام به یک راس معمولی که در پایین آن قرار گرفته است متصل می‌شود. با فرض اینکه چندضلعی از قبل ذوزنقه‌ای شده است، هزینه کلی الگوریتم از مرتبه O(K) است.

الگوریتم (۱)

```
HandleMergeVertex (v)
Input: a split vertex v
Found ← false, i ← 0
t ← AdjacentTrapezoid(v)
while ((not Found) and (i < K) and (t ≠ null))
    VisiblePortion ← ComputeVisiblePortion(t)
    if (Length(VisiblePortion) == 0)
        exit while
    if (t has only 1 lower adjacent trapezoid)
        t ← GetLowerTrapezoid(t)
    else if (t has 2 lower adjacent trapezoids)
        SplitVertex ← GetSplitVertex(t)
        if (VisiblePortion.X1 ≤ SplitVertex.X ≤
        VisiblePortion.X2)
            Found ← true
            AddDiagonal (v, SplitVertex)
        else if (VisiblePortion.X1 < SplitVertex.X)
            t ← BottomLeftTrapezoid(t)
        else
            t ← BottomRightTrapezoid(t)
    i ← i + 1
if (not Found)
    AddDiagonal(v,
    BottomVertex(AdjacentTrapezoid(v)))
```

قطری که توسط رویه HandleMergeVertex به چندضلعی اضافه می‌شود، تمامی ذوزنقه‌هایی که در مسیر آن واقع شده است را به دو قسمت تقسیم می‌کند. شکل (۳) یک چندضلعی را قبل و بعد از اجرای رویه HandleMergeVertex نشان می‌دهد.



شکل (۳): قبل و بعد از فراخوانی رویه HandleMergeVertex برای یک راس تفکیک

برای اتصال یک راس تفکیک به یک راس ادغام می‌توان مشابه با الگوریتم (۱) عمل کرد. تنها تفاوت موجود این است که می‌بایست زنجیره ذوزنقه‌ها به سمت بالا پیمایش شود و بازه آشکار روی قاعده بالایی هر ذوزنقه محاسبه شود.

۴-۲- الگوریتم پیشنهادی

با استفاده از روشی که برای حذف راس‌های ادغام و تفکیک ارائه شد، می‌توان یک چندضلعی ساده و یا حفره‌دار را به صورت یکنواخت تجزیه کرد. برای انجام این کار ابتدا باید چندضلعی به صورت ذوزنقه‌ای تجزیه شود. الگوریتم‌های کارآیی برای تجزیه ذوزنقه‌ای یک چندضلعی ساده و یا حفره‌دار وجود دارد [۷،۶،۵]. چنانچه از الگوریتم‌هایی که چندضلعی‌های حفره‌دار را به صورت ذوزنقه‌ای تجزیه می‌کنند استفاده شود، قادر به تجزیه یکنواخت یک چندضلعی حفره‌دار خواهیم بود.

در الگوریتم پیشنهادی ما، رویه HandleMergeVertex به ازای هر راس ادغام فراخوانی می‌شود. بعد از پردازش همه راس‌های ادغام، راس‌های تفکیک پردازش می‌شوند. برای از بین بردن راس‌های تفکیک، نیازی به پیمایش ذوزنقه‌ها به سمت بالا نیست زیرا در چندضلعی راس ادغامی وجود ندارد (راس‌های ادغام قبلا از بین رفته‌اند). بنابراین کافی است راس‌های تفکیک را به نزدیک‌ترین راسی که بالای آنها واقع شده است متصل کرد.

همواره می‌توان هر راس تفکیک را به یکی از راس‌های قاعده بالایی (راس سمت چپ و یا سمت راست) ذوزنقه مجاور آن متصل کرد. این کار در O(1) انجام می‌گیرد. برای انجام این کار فرض می‌کنیم رویه‌ای به نام HandleSplitVertex داریم که در زمان O(1)

یک راس تفکیک را به یک راس معمولی که بالای آن واقع شده است متصل می‌کند. الگوریتم (۲) روش پیشنهادی ما را برای تجزیه یکنواخت یک چندضلعی نشان می‌دهد.

الگوریتم (۲)

Algorithm GreedyMonotonDecomposition
Input: a polygon *P* with *n* vertices(with or without holes)
Output: a set of diagonals that decompose *P* into Y-monotone polygons.

1. Compute horizontal trapezoidal Decomposition of *P*
2. Find all split vertices of *P* and put them in array *S*.
3. Find all merge vertices of *P* and put them in array *M*.
4. **for** each merge vertex *v* in array *M* **do**
   call HandleMergeVertex(*v*)
5. **for** each split vertex *v* in array *S* **do**
   if *v* is not an endpoint of a diagonal **then**
   call HandleSplitVertex(*v*)

در گام اول چندضلعی به صورت ذوزنقه‌ای تجزیه می‌شود. زمان لازم برای اجرای این گام را با $O(T)$ نمایش می‌دهیم. گام دوم و سوم هر کدام در مدت زمان $O(n)$ اجرا می‌شوند. مدت زمان اجرای رویه HandleMergeVertex در بدترین حالت از مرتبه $O(K)$ است. اگر تعداد راس‌های ادغام برابر N_m و تعداد راس‌های تفکیک چندضلعی برابر N_s باشد، زمان اجرای گام چهارم از مرتبه $O(K\times N_m)$ و گام پنجم از مرتبه $O(N_s)$ خواهد بود. بنابراین زمان کل اجرای الگوریتم پیشنهادی از رابطه (۱) بدست می‌آید:

$$t(n, N_m, N_s) \in O(T) + O(n) + O(K\times N_m) + O(N_s) \qquad (۱)$$

با توجه به اینکه $N_s \in O(n)$ و همچنین تجزیه ذوزنقه‌ای یک چندضلعی با n راس از مرتبه $\Omega(n)$ است، رابطه (۱) را می‌توان به صورت زیر بازنویسی کرد:

$$t(n, N_m) \in O(T) + O(K\times N_m) \qquad (۲)$$

اگر K یک مقدار ثابت در نظر گرفته شود (مستقل از مقدار n) آنگاه:

$$t(n, N_m) \in O(T) + O(N_m) \qquad (۳)$$

با توجه به اینکه $N_m < n$ است، در نتیجه خواهیم داشت:

$$t(n, N_m) \in O(T) \qquad (۴)$$

بنابراین اگر K یک مقدار ثابت باشد، هزینه کلی الگوریتم برابر با هزینه تجزیه ذوزنقه‌ای چندضلعی P است. اگر مقدار K متناسب با مقدار n انتخاب شود، یعنی $K\in O(n)$ باشد، آنگاه هزینه کلی الگوریتم (۲) از مرتبه زیر است:

$$t(n, N_m) \in O(T) + O(n\times N_m) \qquad (۵)$$

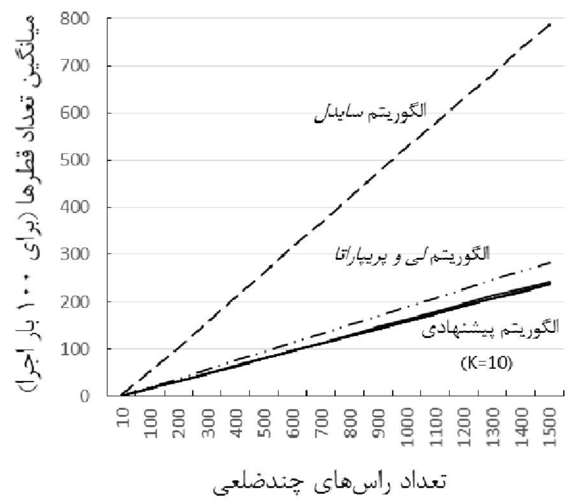
۴-۳- پیاده‌سازی و مقایسه با الگوریتم‌های موجود

برای پیاده‌سازی الگوریتم ارائه شده، ابتدا باید از الگوریتم مناسبی جهت تجزیه ذوزنقه‌ای چندضلعی استفاده کرد. برای این منظور از نسخه اولیه الگوریتم تصادفی افزایشی[9] *سایدل* استفاده کرده‌ایم [۵]. زمان اجرای مورد انتظار[10] این الگوریتم از مرتبه $O(n \log n)$ است. در بخش قبل ثابت کردیم که زمان اجرای الگوریتم حریصانه برای حالتی که حداکثر عمق جستجو ثابت است، برابر با زمان مورد نیاز برای تجزیه ذوزنقه‌ای چندضلعی است. با توجه به اینکه در آزمایش‌ها حداکثر عمق جستجو را مقداری ثابت در نظر گرفته‌ایم، زمان اجرای الگوریتم پیاده‌سازی شده از مرتبه $O(n \log n)$ است. نتایج پیاده‌سازی نشان می‌دهد که در عمل نیازی به استفاده از مقادیر بزرگ به عنوان حداکثر عمق جستجو نیست (حتی برای چندضلعی‌هایی که بیش از ۱۰۰۰ راس دارند با $K=۱۰$ نتایج مطلوبی حاصل شده است).

در نمودار (۱) نتایج حاصل از مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی* و *پریپاراتا* رسم شده است. برای مقایسه این الگوریتم‌ها با یکدیگر از یک پایگاه داده حاوی ۱۶۰۰ چندضلعی ساده که به صورت تصادفی تولید شده‌اند استفاده شده است. محور x نمودار (۱) تعداد راس‌های چندضلعی‌های تولید شده را نشان می‌دهد. به ازای هر مقدار مانند v روی محور x، یک مجموعه با ۱۰۰ عدد چندضلعی ساده که هر کدام از آنها v راس دارد تولید شده است. محور y میانگین تعداد قطرهایی که هر الگوریتم برای تجزیه این مجموعه از چندضلعی‌ها بکار برده است را نشان می‌دهد (مجموع تعداد قطرهای بکار رفته برای تجزیه ۱۰۰ چندضلعی، تقسیم بر عدد ۱۰۰).

نتایج پیاده‌سازی نشان می‌دهد که در عمل با استفاده از مقادیر کوچک K می‌توان به نتایج مطلوبی دست یافت. این نتایج نشان می‌دهد که در عمل الگوریتم پیشنهادی از تعداد قطرهای بسیار کمتری نسبت به الگوریتم *سایدل* استفاده می‌کند (به طور میانگین کاهش ۷۰ درصدی در استفاده از قطرها برای تجزیه چندضلعی‌هایی با ۱۵۰۰ راس). این الگوریتم نسبت به الگوریتم *لی* نیز از تعداد قطرهای کمتری برای تجزیه چندضلعی استفاده می‌کند (به طور میانگین کاهش ۱۷ درصدی در استفاده از قطرها برای تجزیه چندضلعی‌هایی با ۱۵۰۰ راس).

نمودار(۱): مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و *لی*



۵- نتیجه

تا کنون پنج الگوریتم برای تجزیه یکنواخت چندضلعی‌های ساده و یا حفره‌دار ارائه شده است [۴]. الگوریتم‌های *لیو* و *نتافوس*، *وی* و *کیل* چندضلعی را به صورت کمینه تجزیه می‌کنند. الگوریتم‌های *لی* و *سایدل* تجزیه را الزاما به صورت کمینه انجام نمی‌دهند. زمان اجرای الگوریتم *کیل* از مرتبه $O(Nn^4)$ و الگوریتم *لیو* و *نتافوس* از مرتبه $O(nN^3 + N^2 n \log n + N^5)$ است. این زمان اجرا در کاربردهای عملی چندان مطلوب نیست. الگوریتم *وی* از زمان اجرای قابل قبولی برخوردار است اما این الگوریتم برای تجزیه چندضلعی از نقاط کمکی استفاده می‌کند. هدف از الگوریتمی که در این مقاله ارائه شده است، بوجود آوردن تعادلی بین زمان اجرا و تجزیه کمینه است. الگوریتم پیشنهادی ما از نقاط کمکی استفاده نمی‌کند. با توجه به اینکه برای تجزیه ذوزنقه‌ای یک چندضلعی، الگوریتم‌های کارآیی موجود است، می‌توان به سرعت یک چندضلعی ساده و یا حفره‌دار را توسط الگوریتم پیشنهادی به صورت یکنواخت تجزیه کرد.

مراجع

[1] Berg, M.D., Cheong, O., Kreveld, M.V. and Overmars, M. 2008. Computational Geometry: Algorithms and Applications, Springer.
[2] Liu, R. and Ntafos, S. 1988. On decomposing polygons into uniformly monotone parts. Information Processing Letters, 27: 85-89.
[3] Keil, J.M. 1996. Polygon Decomposition. Department of Computer Science, University of Saskatchewan, Saskatoon Sask, Canada.
[4] Wei, X., Joneja, A. and Mount, D.M. 2012. Optimal uniformly monotone partitioning of polygons with holes. Computer-Aided Design, 44: 1235–1252.
[5] Seidel, R. 1991. A simple and fast incremental randomized algorithm for computing trapezoidal and for triangulating polygons. Computational Geometry: Theory and Applications, 1:51-64.
[6] Lorenzetto, G.P. and Datta, A. 2002. A linear time heuristics for trapezoidation of GIS polygons. ICCS, LNCS 2331, 75−84.
[7] Zalik, B., Clapworthy, G.J. 1999. A universal trapezoidation algorithm for planar polygons. Computers & Graphics, 23: 353-363.

زیرنویس‌ها

1 Monotone
2 Minimum ink decomposition
3 Greedy
4 Steiner points
5 Visibility graph
6 Y-monotone polygon
7 Turn vertex
8 Visible
9 Incremental randomized algorithm
10 Expected running time