دانشگاه تبریز
دانشکده ریاضی
گروه علوم کامپیوتر

پایان‌نامه

برای دریافت درجه کارشناسی ارشد در رشته علوم کامپیوتر

عنوان

ارائه یک الگوریتم جهت تجزیه یکنواخت چندضلعی‌های ساده در فضای دوبعدی

استاد راهنما
دکتر جابر کریم‌پور

استاد مشاور
دکتر شهریار لطفی

پژوهشگر
احمد تاجدینی

شهریور ۱۳۹۲

بسم الله الرحمن الرحيم

نام خانوادگی: تاجدینی نام: احمد

عنوان پایان‌نامه: ارائه یک الگوریتم جهت تجزیه یکنواخت چندضلعی‌های ساده درفضای دوبعدی

استاد راهنما: دکتر جابر کریم پور

استاد مشاور: دکتر شهریار لطفی

مقطع تحصیلی: کارشناسی ارشد رشته: علوم کامپیوتر گرایش: سیستم‌های کامپیوتری دانشگاه: تبریز

دانشکده: ریاضی	تاریخ فارغ‌التحصیلی: ۱۳۹۲/۶/۲۰	تعداد صفحات: ۶۵

کلید واژه‌ها: تجزیه چندضلعی، چندضلعی یکنواخت، چندضلعی ساده، چندضلعی حفره‌دار، الگوریتم حریصانه، هندسه محاسباتی

چکیده:

به افراز یک چندضلعی به مجموعه‌ای از چندضلعی‌های کوچکتر، تجزیه چندضلعی گفته می‌شود. یک چندضلعی را می‌توان به روش‌های مختلفی تجزیه کرد. از میان این روش‌ها، تجزیه محدب، یکنواخت و ذوزنقه‌ای بیشترین کاربردها را در حوزه هندسه محاسباتی دارند. در برخی مواقع، لازم و یا مطلوب است تجزیه به گونه‌ای انجام شود که با حفظ قیدهای مسئله، تعداد چندضلعی‌های تولید شده کمینه باشد. تجزیه چندضلعی‌ها در حوزه‌های مختلفی مانند گرافیک برداری، تشخیص الگو، تشخیص متن، محاسبه جمع‌های مینکوفسکی و طرح‌ریزی حرکت ربات کاربرد دارد.

در این پایان‌نامه به ارائه یک الگوریتم حریصانه جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته می‌شود. این الگوریتم بدون استفاده از نقاط کمکی، یک چندضلعی را به صورت یکنواخت تجزیه می‌کند. هدف اصلی از ارائه این الگوریتم، دستیابی به تجزیه‌ای «تقریبا کمینه» در یک زمان قابل قبول است. از آنجا که 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 الگوریتم پیشنهادی، دو مقوله مورد توجه قرار گرفته است. مقوله اول کمینگی تجزیه است. این الگوریتم 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. دومین مقوله که در طراحی این الگوریتم مورد توجه قرار گرفته است، زمان اجرای آن است. بخشی از زمان اجرای این الگوریتم با استفاده از پارامتری به نام «حداکثر عمق جستجو» کنترل می‌شود. هر چه مقدار این پارامتر کوچکتر باشد، احتمال یافتن تجزیه‌ای کمینه یا تقریبا کمینه کمتر می‌شود اما زمان اجرای الگوریتم نیز به همان نسبت کاهش می‌یابد. با انتساب مقادیر بزرگتر به این پارامتر، می‌توان جواب‌های بهتری تولید کرد اما زمان اجرای الگوریتم نیز افزایش می‌یابد. با تنظیم مقدار این پارامتر و با توجه به کاربرد مسئله، می‌توان از بین زمان اجرا و کمینه بودن تجزیه یکی را ترجیح داد و یا اینکه تعادلی بین آنها بوجود آورد.

به پاس تعبیر زیبا و انسانی شان از کلمه ایثار و از خودگذشتگی

به پاس عاطفه سرشار و گرمای امید بخش وجودشان

به پاس قلب های بزرگشان که فریادرس است و سرگردانی و ترس در پناهشان به شجاعت می گراید

و به پاس محبت‌های بی دریغشان که هرگز فروکش نمی کند

این مجموعه را تقدیم می کنم به:

پدر و مادر بزرگوار و مهربانم

و

خواهر و دو برادر عزیزم

و با سپاس از زحمات اساتید راهنما و مشاور گرامیم

جناب آقای دکتر کریم پور و جناب آقای دکتر لطفی

چرا که بدون راهنمایی‌های ارزنده ایشان تهیه این پایان نامه بسیار مشکل می نمود

فهرست مطالب

عنوان شماره صفحه

فهرست جدول‌ها

عنوان شماره صفحه

فهرست شکل‌ها

عنوان شماره صفحه

فصل ۱

مقدمه

تجزیه چندضلعی[1] 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[2] و یا نسبت به یک خط خاص یکنواخت[3] باشند. در برخی مواقع لازم و یا مطلوب است که تجزیه به گونه‌ای انجام شود که تعداد چندضلعی‌های تولید شده کمینه باشد. در مواردی نیز هدف صرفا تجزیه چندضلعی به مجموعه‌ای از چندضلعی‌های کوچکتر است و الزامی وجود ندارد که چندضلعی‌های حاصل از افراز ویژگی خاصی داشته باشند. تجزیه چندضلعی‌ها در حوزه‌های مختلفی مانند گرافیک برداری، تشخیص الگو، تشخیص متن، محاسبه جمع‌های مینکوفسکی و طرح‌ریزی حرکت ربات[4] کاربرد دارد. از آنجایی که اعمال کردن اکثر الگوریتم‌ها روی چندضلعی‌های محدب یا یکنواخت، ساده‌تر و کم‌هزینه‌تر 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 پایان‌نامه به ارائه یک الگوریتم حریصانه جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته می‌شود. این الگوریتم از نقاط کمکی[5] برای تجزیه چندضلعی استفاده نمی‌کند.

[1] polygon decomposition
[2] convex
[3] monotone with respect to a line
[4] robot motion planning
[5] Steiner points

۱-۱ اصطلاحات

اصطلاحات بکار رفته در این پایان‌نامه در جدول ۱-۱ ملاحظه می‌شود. اکثر این اصطلاحات در حوزه هندسه محاسباتی شناخته شده هستند اما برخی از آنها تنها در این پایان‌نامه رایج می‌باشند. این اصطلاحات جدید با علامت ستاره مشخص شده‌اند.

جدول ۱-۱) اصطلاحات بکار رفته در این پایان‌نامه

تجزیه چندضلعی	افراز یک چندضلعی به مجموعه‌ای از چندضلعی‌های کوچکتر، تجزیه چندضلعی نامیده می‌شود.
تجزیه کمینه	تجزیه‌ای که تعداد چندضلعی‌های حاصل از آن کمینه است.
تجزیه کمینه با کمترین میزان مصرف جوهر	تجزیه‌ای که در آن مجموع طول قطرهای اضافه شده به چندضلعی، کمینه است.
چندضلعی حفره دار	چندضلعی که درون آن، نواحی تو خالی (به شکل چندضلعی) وجود دارد.
زیر فضا	یک افراز از فضای دو و یا سه بعدی
قطر (در چندضلعی)	پاره‌خطی که دو راس غیر مجاور چندضلعی را به هم متصل می‌کند.
راس ادغام	راسی که زاویه داخلی آن بیشتر از ۱۸۰ درجه است و پایین‌تر از راس‌های مجاور خود واقع شده است.
راس تفکیک	راسی که زاویه داخلی آن بیشتر از ۱۸۰ درجه است و بالاتر از راس‌های مجاور خود واقع شده است.
قطر ویژه*	قطری که یک انتهای آن به راس ادغام و انتهای دیگر آن به راس تفکیک متصل شده باشد.
قطر معمولی*	قطری که فقط یک انتهای آن به راس ادغام و یا تفکیک متصل باشد.
قطر غیر مفید*	قطری که هیچ یک از دو انتهای آن به راس ادغام یا تفکیک متصل نباشد.
از بین بردن راس تفکیک (ادغام) *	تبدیل کردن یک راس تفکیک (ادغام) به یک راس معمولی بوسیله اضافه کردن یک قطر به آن.
مثلثی‌سازی	تجزیه یک چندضلعی به مجموعه‌ای از مثلث‌ها.

۱-۲ بیان مسئله

در این تحقیق به مطالعه و بررسی الگوریتم‌های موجود و همچنین ارائه یک الگوریتم جهت تجزیه یکنواخت[1] چندضلعی‌های ساده و حفره‌دار پرداخته می‌شود. در این مسئله، می‌بایست یک چندضلعی ساده و یا حفره‌دار به گونه‌ای تجزیه شود که همه چندضلعی‌های حاصل شده از آن، نسبت به محور Y یکنواخت باشند. به عبارت دیگر همه چندضلعی‌ها باید به صورت یکپارچه نسبت به محور Y یکنواخت باشند[2]. با توجه به شرایط و محدودیت‌های مسئله، در مورد مجاز بودن در به کارگیری نقاط کمکی تصمیم‌گیری می‌شود. استفاده از نقاط کمکی باعث افزایش تعداد راس‌های چندضلعی اولیه می‌شود و بهتر است تا جایی که امکان دارد، استفاده از این نقاط محدود شود.

۱-۳ نظریه

تجزیه کمینه یکنواخت یک چندضلعی حفره‌دار، هنگامی که استفاده از نقاط کمکی مجاز نیست، یک مسئله NP-سخت است. این مسئله را به اختصار MMD می‌نامیم[3]. در حالتی که استفاده از نقاط کمکی مجاز است، الگوریتم‌هایی با زمان اجرای چندجمله‌ای برای حل این مسئله وجود دارد اما زمان اجرای آنها مطلوب نیست. در این تحقیق سعی می‌کنیم به این پرسش پاسخ دهیم که آیا می‌توان با ارائه یک الگوریتم حریصانه برای مسئله MMD به تجزیه‌ای تقریبا کمینه دست یافت و یا خیر. کارایی این الگوریتم را می‌توان با استفاده از پیاده‌سازی و مقایسه با برخی از الگوریتم‌های موجود برای تجزیه یکنواخت چندضلعی، مورد بررسی و تحلیل قرار داد.

[1] monotone decomposition
[2] uniformly monotone with respect to axis Y
[3] Minimum Monotone Decomposition

۱-۴ اهداف پایان‌نامه

هدف از این تحقیق، ارائه یک الگوریتم جهت حل مسئله «تجزیه چندضلعی‌های ساده و یا حفره‌دار به چندضلعی‌های یکنواخت قائم» است. تا کنون الگوریتم‌های مختلفی برای تجزیه چندضلعی‌های ساده و حفره‌دار ارائه شده است. برخی از الگوریتم‌های ارائه شده، چندضلعی را به صورت کمینه تجزیه می‌کنند به این معنا که تعداد چندضلعی‌های حاصل شده از فرآیند تجزیه، کمینه است. تجزیه کمینه یک چندضلعی حفره‌دار، بدون استفاده از نقاط کمکی، یک مسئله NP-سخت است. زمان اجرای الگوریتم‌های موجود برای تجزیه کمینه یک چندضلعی ساده، بدون استفاده از نقاط کمکی، چندان مطلوب نیست. در این تحقیق به توسعه الگوریتمی پرداخته می‌شود که به صورت *حریصانه* تلاش می‌کند یک چندضلعی ساده و یا حفره‌دار را بدون استفاده از نقاط کمکی به صورت یکنواخت تجزیه کند. هدف اصلی این الگوریتم، دستیابی به یک تجزیه تقریبا کمینه (و حتی در برخی مواقع کمینه) در یک زمان قابل قبول است.

۱-۵ سازمان پایان‌نامه

پس از مقدمه، در فصل ۲ به تشریح برخی از الگوریتم‌های موجود برای تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته می‌شود. این فصل به سه بخش تقسیم شده است. در بخش اول به برخی تعاریف اولیه و مفاهیم مقدماتی از هندسه محاسباتی اشاره شده است. این مفاهیم و تعاریف اولیه در سرتاسر این پایان‌نامه کاربرد دارند. در بخش دوم الگوریتم‌های لی و پریپاراتا، لیو و سایدل برای تجزیه یکنواخت یک چندضلعی تشریح شده است. این الگوریتم‌ها بدون استفاده از نقاط کمکی، یک چندضلعی را به صورت یکنواخت تجزیه می‌کنند. در بخش سوم به مقایسه الگوریتم‌های موجود و نتیجه‌گیری در مورد آنها پرداخته شده است.

در فصل سوم به ارائه یک الگوریتم جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته شده است. این فصل شامل سه بخش است. در بخش اول یک الگوریتم اولیه برای حل این مسئله ارائه می‌شود. در بخش دوم به بهبود عملکرد و زمان اجرای الگوریتم اولیه پرداخته می‌شود. نتایج حاصل از پیاده‌سازی و مقایسه الگوریتم پیشنهادی با برخی از الگوریتم‌های موجود، در بخش سوم از فصل ۳ بحث و بررسی شده است.

در فصل چهارم به نتیجه‌گیری کلی و تشریح روند ادامه این تحقیق پرداخته شده است. در این فصل ضمن مرور مطالب تشریح شده در فصل‌های قبل و نتیجه‌گیری کلی در مورد آنها، پیشنهاداتی برای بهبود عملکرد الگوریتم ارائه شده در این پایان‌نامه مطرح شده است.

فصل ۲

پیشینه‌های پژوهشی

تجزیه چندضلعی‌ها یکی از مباحث مهم در حوزه هندسه محاسباتی است. تا به حال الگوریتم‌های متعددی برای تجزیه یک چندضلعی به چندضلعی‌های ساده[1]، یکنواخت[2]، محدب[3]، ستاره‌ای[4] و ذوزنقه‌ای[5] ارائه شده است [۴،۳،۲،۱]. در بسیاری از موارد، تجزیه یک چندضلعی، گامی از مراحل اجرای یک الگوریتم دیگر است. از میان انواع مختلف تجزیه، تجزیه به چندضلعی‌های یکنواخت و محدب و ذوزنقه‌ای مورد توجه بیشتری قرار گرفته است [۴]. دلیل این موضوع این است که بسیاری از الگوریتم‌ها روی این نوع چندضلعی‌ها به صورت کارآتری اجرا می‌شوند [۶،۵،۳،۱].

در برخی از کاربردها، لازم و یا مطلوب است که تجزیه به صورت کمینه انجام شود [۵،۴]. 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[6] گفته می‌شود. در این تحقیق، منظور از کمینه بودن، کمینه بودن تعداد چندضلعی‌های حاصل از تجزیه است.

پیچیدگی محاسباتی اکثر الگوریتم‌های ارائه شده برای تجزیه چندضلعی، معمولا تحت تاثیر سه عامل قرار دارد. عامل اول تعداد راس‌های چندضلعی است. عامل دوم تعداد راس‌هایی از چندضلعی است

[1] simple
[2] uniform
[3] convex
[4] star-shaped
[5] trapezoidal
[6] minimum ink decomposition

که مقدار زاویه داخلی آنها بیشتر از ۱۸۰ درجه است. به این راس‌ها، راس بازتابی[1] یا راس شکافی[2] گفته می‌شود. عامل سوم تعداد حفره‌های[3] چندضلعی است. در سرتاسر این پایان‌نامه، 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[4] 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 که از این نقاط استفاده نشود یا اینکه میزان استفاده از آنها محدود شود.

برای تجزیه یک چندضلعی ساده، *لی* و *پریپاراتا*[5] یک الگوریتم با زمان اجرای $O(n \log n)$ ارائه داده‌اند. 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*[6] 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 [۵،۴]. چنانچه استفاده از نقاط کمکی مجاز باشد، الگوریتم لیو و نتافوس (با اعمال تغییراتی) به یک الگوریتم با پیچیدگی محاسباتی $O(n N^3 \log n + N^5)$ 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.

مسئله تجزیه یک چندضلعی ساده، حالت خاصی از مسئله تجزیه چندضلعی‌های حفره‌دار است (حالتی که $h=0$ است). *وی*[7] یک الگوریتم با زمان اجرای $O(K(n \log n + h\log^3 h))$ برای تجزیه

---

[1] reflex vertex
[2] notch vertex
[3] hole
[4] steiner points
[5] Lee and Preparata
[6] M. Keil
[7] Wei

کمینه یک چندضلعی حفره‌دار با استفاده از نقاط کمکی، ارائه کرده است [۲۷]. در عبارت اخیر، K تعداد اضلاع گراف آشکاری[1] چندضلعی را نشان می‌دهد. *کیل* ثابت کرد که مسئله تجزیه کمینه یک چندضلعی حفره‌دار به چندضلعی‌های یکنواخت، هنگامی که استفاده از نقاط کمکی مجاز نیست، یک مسئله NP-سخت است [۴].

در این فصل، بعد از مرور برخی تعاریف و مفاهیم مقدماتی مورد نیاز، الگوریتم‌های تجزیه لی و پریپاراتا[2]، لیو و نتافوس و سایدل مورد مطالعه و بررسی قرار می‌گیرد. این الگوریتم‌ها یک چندضلعی را به مجموعه‌ای از چندضلعی‌های یکنواخت قائم تجزیه می‌کنند.

۲-۱ تعاریف اولیه و مفاهیم مقدماتی

در این بخش به تعاریف چندضلعی‌های ساده، محدب و یکنواخت اشاره شده است. همچنین به بررسی نحوه نمایش یک چندضلعی ساده در حافظه و انواع راس‌های یک چندضلعی پرداخته می‌شود. قضیه چندضلعی یکنواخت قائم نقش کلیدی در همه الگوریتم‌های تجزیه یکنواخت ایفا می‌کند. این قضیه در انتهای این بخش معرفی و اثبات می‌شود.

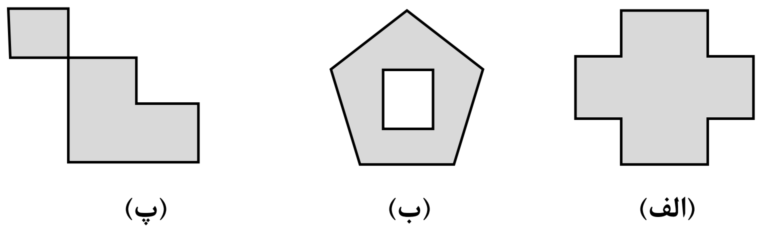
[1] visibility graph
[2] Lee and Preparata

۲-۱-۱ انواع چندضلعی

در هندسه، چندضلعی به زیرمجموعه‌ای از فضای دو بعدی گفته می‌شود که با مسیری بسته شامل تعداد متناهی خطوط راست، محیط شده باشد. در تعریفی ساده‌تر، به هر خط شکسته بسته چندضلعی گفته می‌شود[۱]. چندضلعی‌ها با توجه به ویژگی‌هایی که دارند به چندین گروه دسته‌بندی می‌شوند. چندضلعی‌های محدب و یکنواخت بیشترین کاربردهای عملی و نظری را در حوزه هندسه محاسباتی دارند [۴،۱]. در ادامه به تعریف چندضلعی ساده، محدب و یکنواخت پرداخته می‌شود.

چندضلعی ساده:

چندضلعی که اضلاع آن یکدیگر را قطع نکنند و دارای حفره نباشد، چندضلعی ساده نامیده می‌شود، در غیر اینصورت آن را چندضلعی غیرساده می‌نامند. شکل ۲-۱ مثال‌هایی از چندضلعی‌های ساده و غیر ساده را نشان می‌دهد.



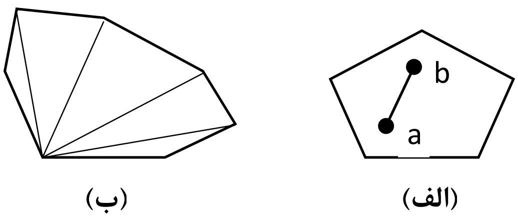
شکل ۲-۱) چند مثال از چندضلعی ساده و غیر ساده. (الف) ساده است. (ب) ساده نیست زیرا حفره دارد. (پ) ساده نیست زیرا اضلاع آن یکدیگر را قطع کرده‌اند.

چندضلعی محدب

چندضلعی که زاویه داخلی همه راس‌های آن کوچکتر و یا مساوی ۱۸۰ درجه است، چندضلعی محدب (کوژ) نامیده می‌شود. اگر دو نقطه دلخواه درون یک چندضلعی محدب با یک پاره‌خط به یکدیگر متصل

[۱] در تعریف اول، چندضلعی به عنوان یک زیرمجموعه از صفحه تعریف شده است. در تعریف دوم، چندضلعی به عنوان یک خط تعریف شده و ناحیه درون آن لحاظ نشده است. در این پایان‌نامه، تفاوت این دو تعریف بی‌اهمیت است.

شوند، آنگاه این پاره‌خط (حتما) درون چندضلعی واقع می‌شود. یک چندضلعی اکیدا محدب نامیده می‌شود اگر و تنها اگر تمامی زاویه‌های داخلی آن کمتر از ۱۸۰ درجه باشند. چندضلعی‌های محدب دارای ویژگی‌های جالبی می‌باشند و در هندسه محاسباتی مورد توجه قرار دارند. یکی از این ویژگی‌ها، سهولت مثلثی‌سازی[1] آنها است. برای مثلثی‌سازی یک چندضلعی محدب کافی است از یکی از راس‌های چندضلعی به تمامی راس‌های غیر مجاور آن قطرهایی رسم شود [1]. اگر تعداد راس‌های چندضلعی n باشد، این عملیات به سادگی در $O(n)$ انجام می‌شود. با توجه به کاربردهای فراوان مثلثی سازی، داشتن چنین الگوریتم ساده و کارایی بسیار مطلوب است.

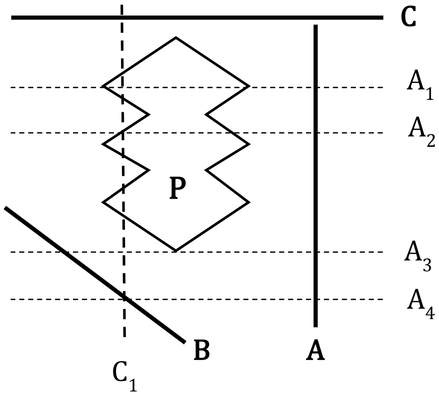


شکل ۲-۲) برخی از ویژگی‌های چندضلعی محدب (الف) پاره‌خط واصل هر دو نقطه درون یک چندضلعی محدب، درون آن واقع می‌شود. (ب) مثلثی‌سازی یک چندضلعی محدب به راحتی صورت می‌گیرد.

چندضلعی یکنواخت

یکنواخت بودن چندضلعی، نسبت به خط تعریف می‌شود. یک چندضلعی ساده نسبت به خط L یکنواخت است[2] اگر و تنها اگر هر خط عمود بر L سطح چندضلعی را حداکثر یک بار قطع کند. به عبارت دیگر یا خط L چندضلعی را قطع نکند و یا اشتراک آن با سطح چندضلعی یک خط پیوسته باشد. چنانچه بتوان خطی عمود بر L رسم کرد که سطح چندضلعی را دو بار و یا بیشتر قطع کند، آن چندضلعی نسبت به خط L یکنواخت نیست. اگر تعدادی چندضلعی نسبت به خط L یکنواخت باشند، آنها را چندضلعی‌های یکپارچه یکنواخت نسبت به خط L می‌نامند[3].

[1] triangulation
[2] monotone with respect line L
[3] uniformly monotone with respect to line L

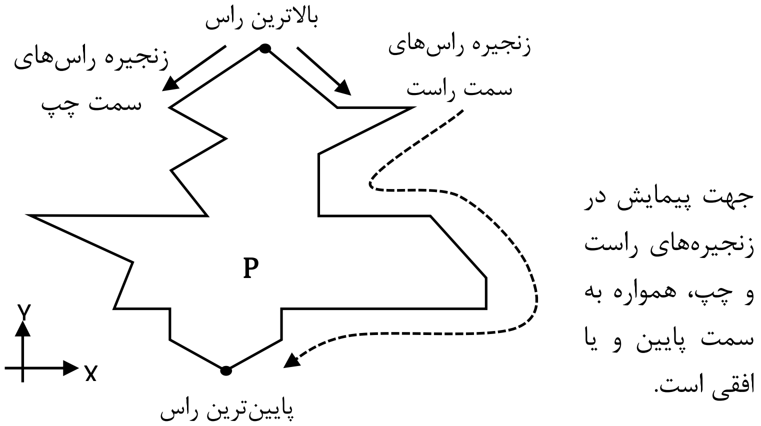


شکل ۲-۳) یک چندضلعی ساده که نسبت به محور Y یکنواخت است

در شکل ۲-۳ یک چندضلعی به نام P رسم شده است. این چندضلعی نسبت به خط A یکنواخت است زیرا هر خطی که عمود بر A رسم می‌شود، یا چندضلعی را قطع نمی‌کند (مانند خط A_4) و یا فقط یک بار سطح چندضلعی را قطع می‌کند (مانند خط‌های A_1، A_2 و A_3). به همین صورت، چندضلعی P نسبت به خط B نیز یکنواخت است. با وجود اینکه P نسبت به خط‌های A و B یکنواخت است اما نسبت به خط C یکنواخت نیست، زیرا اشتراک خط C_1 (که بر C عمود است) و چندضلعی P یک پاره خط پیوسته نیست بلکه از اشتراک این دو، سه پاره‌خط به وجود آمده است (C_1 سه بار سطح P را قطع کرده است).

یک چندضلعی که نسبت به یک خط موازی با محور Y یکنواخت باشد، چندضلعی یکنواخت قائم[1] نامیده می‌شود. یکی از ویژگی‌های چندضلعی‌های یکنواخت قائم این است که اگر راس‌های چندضلعی از بالاترین راس به سمت پایین‌ترین راس پیمایش شود (از سمت زنجیره راس‌های چپ و یا راست) آنگاه جهت پیمایش همواره به سمت پایین و یا افقی است و هیچگاه به سمت بالا نیست. این موضوع در شکل ۲-۴ نشان داده شده است.

[1] Y-monotone polygon



شکل ۲-۴) یک چندضلعی یکنواخت قائم

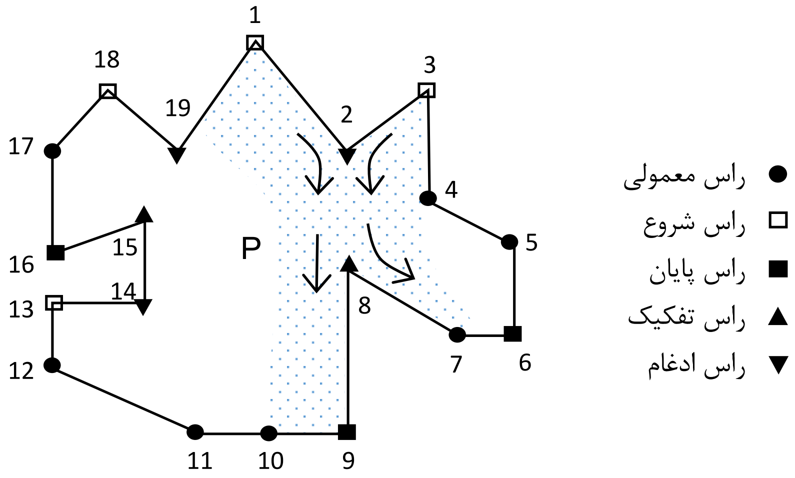
چندضلعی‌های یکنواخت بر خلاف چندضلعی‌های محدب، آزادتر هستند و می‌توانند اشکال متنوع‌تری را اختیار کنند. دلیل این موضوع این است که هر چندضلعی محدب، یک چندضلعی ساده است که نسبت به هر خط دلخواه (در صفحه) یکنواخت است[1]. یکنواخت بودن نسبت به تعداد بی‌شماری خط باعث محدود شدن شکل چندضلعی‌های یکنواخت به اشکال تقریبا حلقوی (دایره‌ای) شکل می‌شود.

۲-۱-۲ انواع راس‌ها در چندضلعی

راس‌های یک چندضلعی با توجه به مقدار زاویه داخلی و همچنین موقعیت نسبی آنها، به پنج دسته تقسیم می‌شوند. شناسایی این دسته‌ها یکی از گام‌های اولیه جهت تجزیه یک چندضلعی ساده است.

در شکل ۲-۵ یک چندضلعی ساده به نام P رسم شده است. به راسی از چندضلعی که بیشترین مقدار مؤلفه Y را داشته باشد، بالاترین راس چندضلعی گفته می‌شود. اگر دو یا چند راس مقدار Y یکسان داشته باشند، راسی بالاتر در نظر گرفته می‌شود که مقدار مؤلفه X آن کمتر باشد.

[1] یک چندضلعی ساده که نسبت به یک خط دلخواه یکنواخت نباشد محدب نیست، زیرا در این صورت نقاطی داخل چندضلعی وجود دارند که پاره‌خط واصل آنها، کاملا داخل چندضلعی واقع نمی‌شود.



شکل ۲-۵) انواع راس‌های یک چندضلعی ساده

در چندضلعی P، راس شماره ۱ بالاترین راس است. همچنین با وجود اینکه مؤلفه Y راس‌های شماره ۶ و ۷ با هم برابر است، اما راس شماره ۷ بالاتر از راس شماره ۶ است زیرا مقدار مؤلفه X آن کمتر است. به راسی که کمترین مقدار مؤلفه Y را داشته باشد، پایین‌ترین راس چندضلعی گفته می‌شود. در چندضلعی P، راس شماره ۹ پایین‌ترین راس چندضلعی است. بالاترین و پایین‌ترین راس چندضلعی توسط دو زنجیره از راس‌ها به هم متصل می‌شوند (زنجیره راس‌های سمت چپ و راست).

یکی از ویژگی‌های چندضلعی یکنواخت قائم این است که اگر راس‌های چندضلعی از بالاترین راس به سمت پایین‌ترین راس پیمایش شود، همواره جهت حرکت به سمت پایین یا افقی خواهد بود و هیچ‌گاه جهت حرکت به سمت بالا تغییر نمی‌کند. چندضلعی P یکنواخت قائم نیست، زیرا در راس‌های شماره ۲ و ۱۹ جهت حرکت از پایین به بالا تغییر کرده است. راسی که در آن جهت حرکت از بالا به پایین و یا از پایین به بالا تغییر می‌کند، راس چرخشی[1] نامیده می‌شود. در چندضلعی P، راس‌های شماره ۱، ۲، ۳، ۶، ۸، ۹، ۱۳، ۱۴، ۱۵، ۱۶، ۱۸ و ۱۹ راس‌های چرخشی هستند.

[1] turn vertex

راسی که چرخشی نباشد، راس معمولی[1] نامیده می‌شود. همواره یکی از راس‌های مجاور یک راس معمولی در بالای آن و راس مجاور دوم نیز در پایین آن قرار می‌گیرد. راس‌های چرخشی به چهار دسته تقسیم می‌شوند:

۱. راس شروع[2]

راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن کمتر از ۱۸۰ درجه است، راس شروع نامیده می‌شود. در چندضلعی P، راس‌های شماره ۱، ۳، ۱۳ و ۱۸ راس‌های شروع می‌باشند.

۲. راس پایان[3]

راسی که از دو راس مجاور خود پایین‌تر و زاویه داخلی آن کمتر از ۱۸۰ درجه است، راس پایان نامیده می‌شود. در چندضلعی P، راس‌های شماره ۶، ۹ و ۱۶ راس‌های پایان می‌باشند.

۳. راس ادغام[4]

راسی که از دو راس مجاور خود پایین‌تر و زاویه داخلی آن بیشتر از ۱۸۰ درجه است، راس ادغام نامیده می‌شود. در چندضلعی P، راس‌های شماره ۲، ۴ و ۱۹ راس‌های ادغام می‌باشند.

۴. راس تفکیک[5]

راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن بیشتر از ۱۸۰ درجه است، راس تفکیک نامیده می‌شود. در چندضلعی P، راس‌های شماره ۸ و ۱۵ راس‌های تفکیک می‌باشند.

علت نامگذاری راس‌های تفکیک به این اسم، این است که این راس‌ها ناحیه داخلی چندضلعی را به دو قسمت تفکیک می‌کنند. برای درک بهتر موضوع، فرض کنید جریان آبی از سمت بالای چندضلعی P (شکل ۲-۵) به سمت پایین آن در حرکت است. جریان آب (فرضی) با نقطه چین و جهت حرکت آن با پیکان مشخص شده است. جریان آب در راس شماره ۸ به دو قسمت تقسیم (تفکیک) می‌شود.

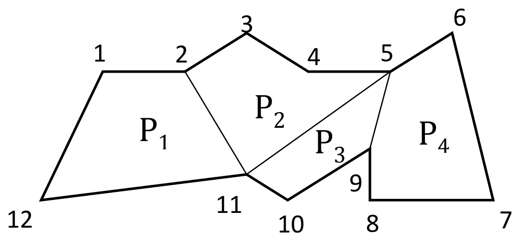
[1] regular Vertex
[2] start Vertex
[3] end Vertex
[4] merge Vertex
[5] split Vertex

به همین صورت، علت نامگذاری راس‌های ادغام این است که اینگونه راس‌ها، دو ناحیه مجاور داخلی را با هم ادغام می‌کنند. برای نشان دادن این موضوع، مجددا فرض کنید جریان آبی از راس‌های شماره ۱ و ۳ تولید می‌شود و به سمت پایین چندضلعی P در حرکت است. این دو جریان مختلف آب، در راس شماره ۲ با هم ادغام می‌شوند.

۲-۱-۳ نمایش چندضلعی و زیر فضا در حافظه

برای پیاده‌سازی الگوریتم‌های هندسه محاسباتی، در ابتدا باید اشکال هندسی را در قالب ساختمان داده‌های[1] مناسب و کارا، در حافظه نمایش داد. از آنجایی که موضوع این تحقیق در رابطه با تجزیه چندضلعی‌ها است، لازم است ساختمان داده مناسبی برای ذخیره‌سازی یک چندضلعی تجزیه شده در حافظه به کار برده شود.

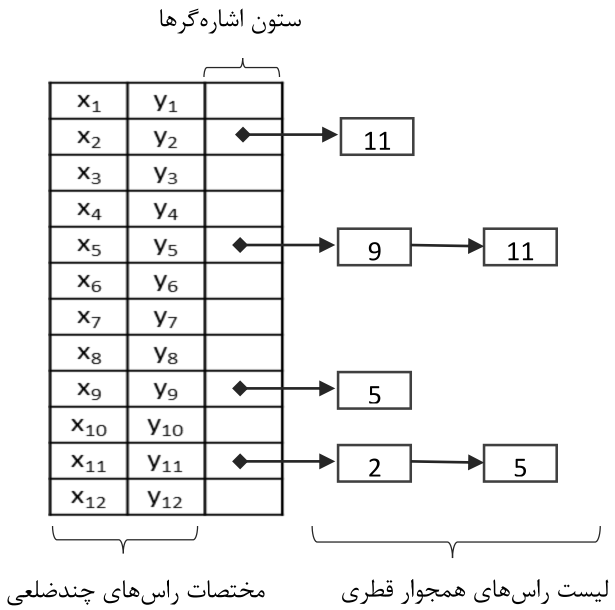
در ساده‌ترین حالت، برای ذخیره کردن یک چندضلعی در حافظه، می‌توان مختصات راس‌های چندضلعی را به ترتیب در یک آرایه یا لیست پیوندی[2] ذخیره کرد. هر چند این روش برای نمایش یک چندضلعی ساده مناسب است، اما قادر به نمایش یک چندضلعی تجزیه شده (زیر فضا[3]) نیست. در یک چندضلعی، هر راس به راس‌های مجاور قبل و بعد از خود متصل است اما در یک چندضلعی تجزیه شده امکان دارد راسی با چندین قطر به راس‌های غیر مجاور متصل شده باشد. شکل ۲-۶ یک چندضلعی تجزیه شده را نشان می‌دهد.



شکل ۲-۶) یک چندضلعی ساده که به چندضلعی‌های یکنواخت قائم تجزیه شده است.

[1] data structures
[2] linked List
[3] subspace

برای نمایش چندضلعی تجزیه شده لازم است هر راس شماره راس‌هایی که با یک قطر به آنها متصل شده است را نگهداری کند. همچنین برای صرفه‌جویی در مصرف حافظه، نیازی به ذخیره اتصال بین دو راس مجاور نیست. این ساختمان داده، تقریبا مشابه با ساختمان داده‌ای است که برای ذخیره‌سازی یک گراف به کار برده می‌شود. شکل ۲-۷ نحوه ذخیره‌سازی چندضلعی ترسیم شده در شکل ۲-۶ را نشان می‌دهد. لیست راس‌های همجوار قطری، راس‌های همجوار با یک راس خاص را نشان می‌دهد که از طریق یک قطر به یکدیگر متصل شده‌اند. قبل از اینکه الگوریتم تجزیه چندضلعی اجرا شود، این لیست خالی است (هیچ قطری وجود ندارد). مختصات هندسی راس‌های چندضلعی، به ترتیب شماره راس، در ستون‌های اول و دوم آرایه ذخیره می‌شوند.



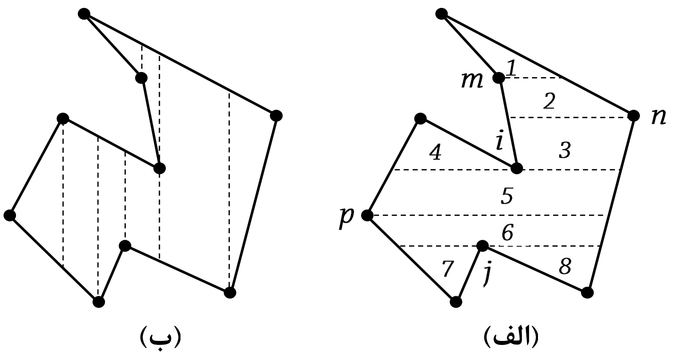
شکل ۲-۷) نمایش چندضلعی شکل ۲-۶ در حافظه

چنانچه چندضلعی شامل حفره باشد، می‌توان از لیست پیوندی و یا آرایه جهت ذخیره‌سازی راس‌های مربوط به حفره‌های چندضلعی استفاده کرد. به منظور ذخیره زیر فضا در حافظه، ساختمان داده‌های کامل‌تری نیز وجود دارد اما برای تجزیه چندضلعی‌ها، ساختمان داده‌ای که تشریح شد کفایت می‌کند.

۲-۱-۴ تجزیه ذوزنقه‌ای یک چندضلعی ساده

یک چندضلعی ساده به نام P را در نظر بگیرید. اگر از هر راس v متعلق به P، به جز راس‌های شروع و پایان، یک پاره‌خط افقی (فرضی) رسم شود و این پاره‌خط تا جایی ادامه پیدا کند که ضلع یا اضلاعی از P را قطع کند، چندضلعی P به مجموعه‌ای از ذوزنقه‌ها و مثلث‌ها تجزیه خواهد شد. این نوع تجزیه چندضلعی، تجزیه ذوزنقه‌ای[1] نامیده می‌شود[2].

اگر پاره‌خط‌ها به صورت عمودی رسم شوند، نوع دیگری از تجزیه ذوزنقه‌ای بدست می‌آید. برای اینکه این دو نوع مختلف تجزیه از یکدیگر متمایز شوند، تجزیه‌ای که با پاره‌خط‌های افقی انجام می‌شود را تجزیه ذوزنقه‌ای افقی و تجزیه‌ای که با پاره‌خط‌های عمودی انجام می‌شود را تجزیه ذوزنقه‌ای عمودی می‌نامیم. شکل ۲-۸ یک چندضلعی که به صورت افقی و عمودی تجزیه شده است را نشان می‌دهد.



شکل ۲-۸) تجزیه ذوزنقه‌ای یک چندضلعی ساده. (الف) تجزیه ذوزنقه‌ای افقی
(ب) تجزیه ذوزنقه‌ای عمودی

چندضلعی شکل ۲-۸ (الف) به صورت افقی تجزیه شده است. پاره‌خط افقی که از راس‌های ادغام و تفکیک رسم می‌شود (راس های i و j) از هر دو سمت چپ و راست تا جایی ادامه پیدا می‌کند که اضلاع

---

[1] trapezoidal decomposition

[2] یک مثلث را می‌توان ذوزنقه‌ای فرض کرد که طول یکی (و تنها یکی) از قاعده‌های آن صفر است. به همین دلیل این نوع تجزیه، تجزیه ذوزنقه‌ای نامیده می‌شود.

چندضلعی را قطع کند. برای راس‌های معمولی (راس‌های m، n و p) پاره خط افقی در یک جهت و به سمت ضلع روبروی آن امتداد پیدا می‌کند. از تقاطع پاره‌خط‌های (فرضی) افقی با اضلاع چندضلعی راس جدیدی بوجود نمی‌آید بلکه آنچه دارای اهمیت است این است که بتوان مختصات چهار گوشه هر ذوزنقه حاصل از تجزیه را محاسبه کرد. ذوزنقه‌هایی که یک راس ادغام روی قاعده بالایی آنها قرار گرفته است (ذوزنقه شماره ۵) دقیقا دارای دو همسایه بالایی می‌باشند (ذوزنقه‌های شماره ۳ و ۴). به همین ترتیب ذوزنقه‌هایی که یک راس تفکیک روی قاعده پایینی آنها قرار گرفته است (ذوزنقه شماره ۶) دقیقا دارای دو همسایه پایینی می‌باشند (ذوزنقه‌های شماره ۷ و ۸).

تجزیه ذوزنقه‌ای در سال ۱۹۸۴ توسط *چازل* و *اینسرپی*[۱] ابداع شد. آنها از این نوع تجزیه برای حل مسئله مثلثی‌سازی استفاده کردند [۶]. تجزیه ذوزنقه‌ای کاربردهای دیگری نیز دارد که از میان آنها می‌توان به برنامه‌ریزی حرکت ربات[۲] و مسئله موقعیت نقطه اشاره[۳] کرد [۱]. تا کنون الگوریتم‌های مختلفی برای تجزیه ذوزنقه‌ای یک چندضلعی ارائه شده است. *سایدل*[۴] یک الگوریتم تصادفی افزایشی[۵] برای حل این مسئله ارائه کرده است. زمان اجرای مورد انتظار[۶] این الگوریتم از مرتبه $O(n \log^* n)$ و پیچیدگی مورد انتظار حافظه مصرفی[۷] آن از مرتبه $O(n)$ است [۷]. نماد $\log^* n$ به صورت بازگشتی تعریف می‌شود. با فرض اینکه $\log^{(i)} n = \log(\log^{(i-1)} n)$ و همچنین $\log^{(0)} n = n$ است، نماد $\log^* n$ بزرگترین عدد صحیح مانند L را نشان می‌دهد به طوری که $\log^{(L)} n \geq 1$.

سایدل نیز از تجزیه ذوزنقه‌ای برای مثلثی سازی یک چندضلعی استفاده کرد. برای انجام این کار ابتدا چندضلعی به صورت ذوزنقه‌ای تجزیه می‌شود و سپس به کمک این تجزیه در زمان خطی به صورت یکنواخت قائم تجزیه می‌شود. در نهایت هر یک از چندضلعی‌های یکنواخت قائم در زمان خطی

[1] Chazelle & Incerpi
[2] robot motion planning
[3] point location problem
[4] Seidel
[5] incremental randomized algorithm
[6] expected running time
[7] expected memory usage

مثلثی‌سازی می‌شوند. با توجه به اینکه تجزیه یکنواخت چندضلعی (گام دوم الگوریتم) و مثلثی‌سازی چندضلعی‌های یکنواخت حاصل از تجزیه (گام سوم الگوریتم) در زمان خطی انجام می‌شود، چندضلعی در زمان $O(n \log^* n)$ مثلثی می‌شود.

برای تجزیه ذوزنقه‌ای یک چندضلعی الگوریتم‌های دیگری نیز ارائه شده است. *لورنزتو* و *داتا* یک الگوریتم تقریبا خطی برای تجزیه چندضلعی ساده و یک الگوریتم از مرتبه $O(n \log n)$ برای تجزیه چندضلعی‌های حفره‌دار ارائه کردند [۸]. *زالیک* و *کلاپورسی* نیز یک الگوریتم با زمان اجرای $O(n^2 \log n)$ برای تجزیه ذوزنقه‌ای یک چندضلعی حفره دار ارائه کرده‌اند [9].

۲-۱-۵ چندضلعی آشکار از یک نقطه

دو نقطه a و b درون یک چندضلعی به نام P مفروض است. می‌گوییم نقاط a و b یکدیگر را می‌بینند[1] اگر و تنها اگر پاره‌خط $\overline{ab}$ به طور کامل درون P واقع شود. به عبارت دیگر:

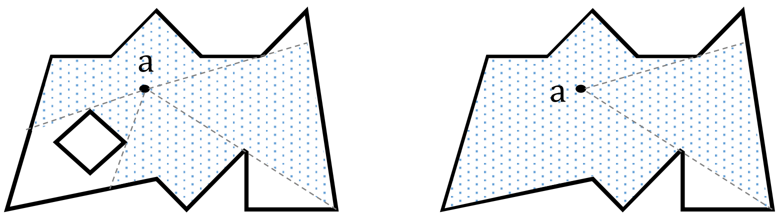
نقاط a و b یکدیگر را می‌بینند $\Leftrightarrow \overline{ab} \cap P = \overline{ab}$

به مجموع تمام نقاط داخل چندضلعی P که از نقطه a آشکار است، چندضلعی آشکار از نقطه a گفته می‌شود[2]. چندضلعی آشکار از یک نقطه، یک چندضلعی ساده است. برای محاسبه چندضلعی آشکار کافی است مختصات راس‌های آن را بدست آورد. در شکل ۲-۹ یک چندضلعی ساده و یک چندضلعی حفره‌دار رسم شده است. چندضلعی آشکار از یک نقطه دلخواه درون این چندضلعی با نقطه‌چین مشخص شده است.

---

[1] یا نقطه a از نقطه b آشکار است (a is visible from b) و برعکس.

[2] visibility polygon from point a



(ب) (الف)

شکل ۲-۹) چندضلعی آشکار از نقطه a در یک چندضلعی ساده (الف) و حفره دار (ب)

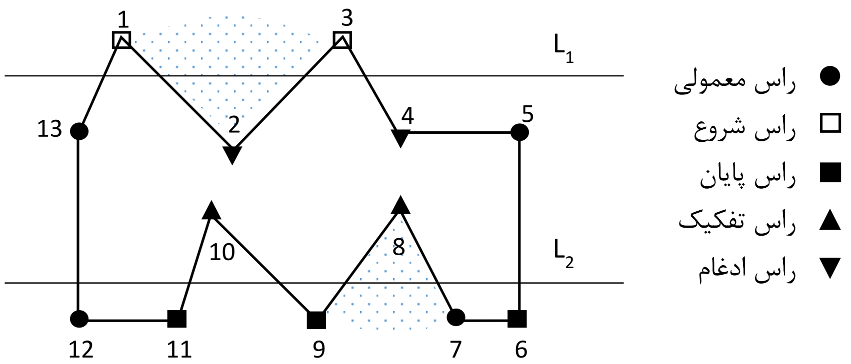
برای محاسبه چندضلعی آشکار الگوریتم‌های متعددی ارائه شده است. برخی از این الگوریتم‌ها چندضلعی آشکار را از یک نقطه محاسبه می‌کند (مانند آنچه در اینجا بررسی شد) و برخی دیگر چندضلعی آشکار را از یک ضلع محاسبه می‌کنند. *چندضلعی آشکار از یک ضلع*[1] مجموع تمام نقاطی از چندضلعی است که از آن ضلع آشکار است. *گیندی*[2] در سال ۱۹۸۱ یک الگوریتم خطی (بهینه) برای محاسبه چندضلعی آشکار از یک نقطه در یک چندضلعی ساده ارائه کرد [۱۰].

دو سال بعد، *لی*[3] نیز یک الگوریتم خطی دیگر برای محاسبه چندضلعی آشکار از یک نقطه (داخل و بیرون چندضلعی) ارائه کرد [۱۱]. الگوریتم‌های خطی دیگری نیز برای این مسئله ارائه شده است [۲۷،۲۶]. از میان الگوریتم‌های مختلفی که برای محاسبه چندضلعی آشکار در چندضلعی‌های حفره‌دار ارائه شده است می‌توان به الگوریتم *آسانو*[4] و الگوریتم *سوری*[5] با زمان اجرای O(n log n) و الگوریتم بهینه *دهنی*[6] با زمان اجرای O(n+h log h) اشاره کرد [۱۴،۱۳،۱۲]. *زارعی* و *قدسی* نیز الگوریتمی برای نسخه بر مبنای پرس و جوی این مسئله[7] ارائه کردند [۱۵].

1 visibility polygon from an edge
2 H. El Gindy
3 D. T. Lee
4 T. Asano
5 S. Suri
6 F. Dehne
7 query based version of the (visibility) problem

۲-۱-۵ قضیه چندضلعی یکنواخت قائم

چندضلعی‌های ساده‌ای که راس ادغام یا تفکیک ندارند، یکنواخت قائم هستند. راس‌های ادغام یا تفکیک در اکثر موارد باعث می‌شوند که چندضلعی، یکنواخت قائم نباشد. این واقعیت را می‌توان به طور شهودی از شکل ۲-۱۰ متوجه شد.



شکل ۲-۱۰) یک چندضلعی ساده که یکنواخت قائم نیست.

راس شماره ۲ یک راس ادغام است. به دلیل اینکه زاویه داخلی این راس بزرگتر از ۱۸۰ درجه است و همچنین دو راس مجاور آن (راس‌های شماره ۱ و ۳) بالای آن قرار گرفته‌اند، یک فضای خالی بالای آن شکل گرفته است. این ناحیه که در شکل با نقطه‌چین مشخص شده است، باعث می‌شود چندضلعی یکنواخت قائم نباشد. به طور مشابه، راس‌های تفکیک نیز دارای چنین ویژگی می‌باشند. راس شماره ۸ یک راس تفکیک است. به دلیل اینکه زاویه داخلی آن بزرگتر از ۱۸۰ درجه است و دو راس مجاور آن (راس‌های شماره ۷ و ۹) در زیر آن قرار گرفته‌اند، یک فضای خالی در زیر این راس شکل گرفته است. این ناحیه باعث می‌شود چندضلعی، یکنواخت قائم نباشد. برای اینکه یک چندضلعی ساده، یکنواخت قائم باشد، کافی است که هیچ راس ادغام یا تفکیکی نداشته باشد. این ادعا در قضیه ۱-۱ ثابت شده است.

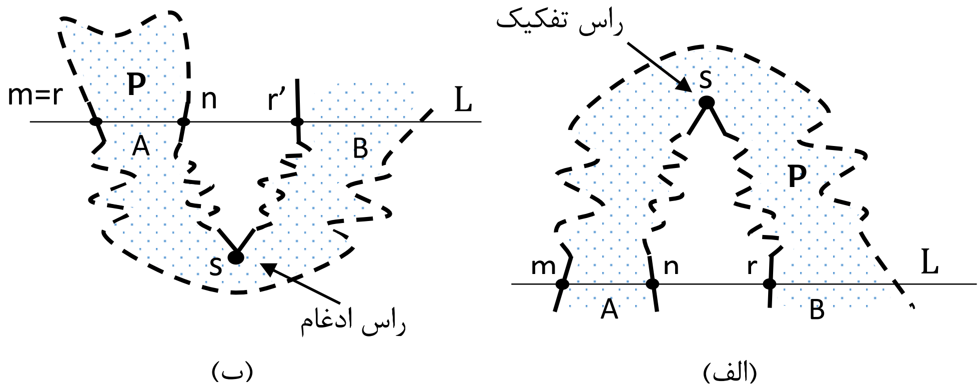
قضیه ۱-۱: قضیه چندضلعی یکنواخت قائم

اگر یک چندضلعی ساده، راس ادغام یا تفکیکی نداشته باشد آنگاه آن چندضلعی، یکنواخت قائم است.

اثبات[1]:

این قضیه را می‌توان با روش اثبات معکوس ثابت کرد. فرض کنید چندضلعی P یکنواخت قائم نیست. باید ثابت کرد که P دارای راس ادغام یا تفکیک است.

از آنجایی که P یکنواخت قائم نیست، لذا خط افقی مانند L وجود دارد که اشتراک آن با P، بیش از یک پاره‌خط را ایجاد می‌کند. مطابق شکل ۲-۱۱ (الف)، دو پاره‌خط اول را a و b نامگذاری می‌کنیم (ممکن است از پاره‌خط‌های بیشتری نیز ایجاد شده باشد). همچنین نقاط انتهایی پاره‌خط A را m و n نامگذاری می‌کنیم.



شکل ۲-۱۱) دو حالت امکان‌پذیر در قضیه چندضلعی یکنواخت قائم

با شروع از راس n، اضلاع چندضلعی P را به نحوی پیمایش می‌کنیم که ناحیه داخلی آن در سمت چپ ما واقع شود (یعنی از راس n به سمت بالا حرکت می‌کنیم). در نقطه‌ای مانند r، خط L دوباره P را قطع می‌کند. اگر $r \neq m$ باشد (شکل ۲-۱۱ الف) آنگاه بالاترین راسی که در پیمایش از n به r ملاقات شده است، یک راس تفکیک است (بنا به تعریف راس تفکیک) و در نتیجه قضیه ثابت می‌شود.

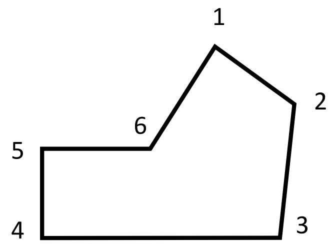
[1] اثبات این قضیه از مرجع [۱] اخذ شده است.

اگر r=m باشد (شکل ۲-۱۱ ب) از راس n شروع به پیمایش اضلاع می‌کنیم، ولی این بار در جهت مخالف (یعنی از راس n به سمت پایین حرکت می‌کنیم). در نقطه‌ای مانند r’، خط L دوباره P را قطع می‌کند. امکان اینکه r’=m باشد وجود ندارد زیرا در اینصورت خط L چندضلعی P را دقیقا در دو ناحیه قطع خواهد کرد و این یک تناقض است (از آنجایی که P یکنواخت قائم نیست، خط L ممکن است P را چندین بار قطع کند و نه فقط دو بار). بنابراین r’≠m است و پایین‌ترین راسی که در پیمایش از n به r’ ملاقات می‌شود یک راس ادغام خواهد بود (بنا به تعریف راس ادغام) و در نتیجه قضیه ثابت می‌شود.

□

قضیه ۲-۱ شرایط کافی برای یکنواخت قائم بودن یک چندضلعی ساده را بیان می‌کند اما باید توجه داشت که این قضیه الزاما در جهت عکس برقرار نیست. اگر یک چندضلعی ساده، یکنواخت قائم باشد، نمی‌توان نتیجه گرفت که این چندضلعی راس تفکیک یا ادغام ندارد.

چندضلعی رسم شده در شکل ۲-۱۲ یک مثال (نقض) است که این ادعا را ثابت می‌کند. این چندضلعی یکنواخت قائم است زیرا اگر راس‌های چندضلعی از بالاترین راس (راس شماره ۱) به سمت پایین‌ترین راس (راس شماره ۳) پیمایش شود (خواه از زنجیره راس‌های سمت چپ یا سمت راست) جهت حرکت همواره رو به پایین یا افقی است. با این حال، راس شماره ۶ یک راس ادغام است.



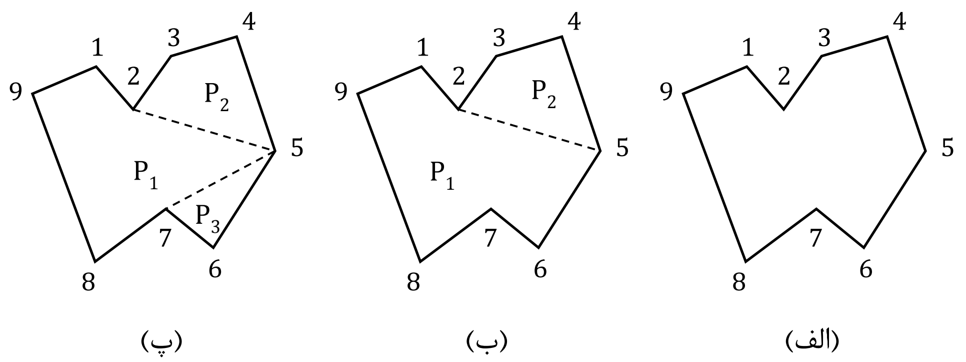
شکل ۲-۱۲) یک مثال برای نشان دادن اینکه قضیه ۱-۱ در جهت عکس برقرار نیست.

۲-۲ الگوریتم لی و پریپاراتا

قضیه ۲-۱ بیان می‌کند برای اینکه یک چندضلعی یکنواخت قائم باشد، کافی است راس ادغام یا تفکیک نداشته باشد. برای تجزیه یک چندضلعی ساده به چندضلعی‌های یکنواخت قائم، کافی است از راس‌های ادغام و تفکیک قطرهایی به راس‌های مناسب (که باید آنها را یافت) اضافه شود. بعد از این فرآیند چندضلعی‌های حاصل از تجزیه بدون راس ادغام یا تفکیک خواهند. باید توجه داشت که قطرهایی که به چندضلعی اضافه می‌شوند، یکدیگر را قطع نکنند زیرا در اینصورت تجزیه معتبر نیست.

لی و پریپاراتا[1] یک الگوریتم با زمان اجرای $O(n \log n)$ برای حل مسئله تجزیه یک چندضلعی ساده ساده به چندضلعی‌های قائم ارائه کردند [۳]. این الگوریتم کرانی را روی تعداد چندضلعی‌های تولید شده قرار نمی‌دهد (تعداد چندضلعی‌های تولید شده الزاما کمینه نیست). در این الگوریتم هر راس ادغام توسط یک قطر به راسی که در پایین آن قرار گرفته است متصل می‌شود. همچنین هر راس تفکیک توسط یک قطر به راسی که در بالای آن قرار گرفته است متصل می‌شود.

بعد از انجام این کار، راس‌های ادغام و یا تفکیک چندضلعی اولیه به راس‌های معمولی تبدیل می‌شوند. شکل ۲-۱۳ مراحل تجزیه یک چندضلعی ساده که با این الگوریتم تجزیه شده است را نشان می‌دهد.



شکل ۲-۱۳) مراحل تجزیه یک چندضلعی ساده توسط الگوریتم *لی* و *پریپاراتا*

[1] Lee and Preparata

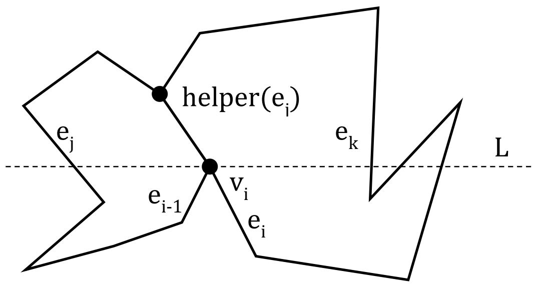
شکل ۲-۱۳ (الف) یک چندضلعی ساده را نمایش می‌دهد. راس شماره ۲ این چندضلعی یک راس ادغام است و بنابراین باید توسط یک قطر به راسی که در پایین آن قرار دارد متصل شود. باید توجه داشت که این قطر، قطرهای دیگر را قطع نکند و همچنین از چندضلعی خارج نشود. راس‌های شماره ۵، ۶، ۷ و ۸ گزینه‌های مناسبی هستند. در اینجا، راس شماره ۵ انتخاب شده است. بعد از اینکه راس‌های شماره ۲ و ۵ با یک قطر به هم متصل شدند، چندضلعی اولیه به دو چندضلعی P_1 و P_2 تجزیه می‌شود. راس شماره ۲ (که در چندضلعی اولیه یک راس ادغام است) در چندضلعی‌های P_1 و P_2 یک راس معمولی است. راس شماره ۷ یک راس تفکیک است و باید توسط یک قطر به راس مناسبی که در بالای آن قرار گرفته متصل شود. راس‌های شماره ۱، ۲، ۵ و ۹ گزینه‌های مناسبی هستند. در اینجا راس شماره ۵ انتخاب شده است. با اضافه کردن قطر دوم، چندضلعی اولیه به سه چندضلعی P_1، P_2 و P_3 تجزیه می‌شود که هیچکدام راس ادغام یا تفکیک ندارند. بنابراین با توجه به قضیه ۱-۱ این چندضلعی‌ها یکنواخت قائم هستند.

برای اضافه کردن قطر به راس‌های ادغام و تفکیک، از روش جاروب صفحه[1] استفاده می‌شود. فرض کنید که v_1، v_2، ... ، v_n راس‌های یک چندضلعی ساده به نام P باشد که به صورت پاد ساعتگرد نامگذاری شده است. همچنین فرض کنید e_1، e_2، ... ، e_n مجموعه اضلاع این چندضلعی باشد به طوری که برای $1 \leq i < n$، $e_i = \overline{v_i\, v_{i+1}}$ و $e_n = \overline{v_n\, v_1}$. در روش جاروب صفحه، یک خط جاروب[2] فرضی به نام L صفحه را از بالا به سمت پایین پویش[3] می‌کند. خط جاروب در نقاط خاصی متوقف می‌شود. به این نقاط، نقاط پردازشی[4] گفته می‌شود. به طور خاص برای این مسئله، خط جاروب روی راس‌های چندضلعی P متوقف می‌شود. در ضمن فرآیند پویش صفحه، نقاط پردازشی جدیدی ساخته نمی‌شود. نقاط پردازشی در یک صف اولویت[5] به نام Q ذخیره می‌شوند. اولویت نقاط داخل صف با توجه به مقدار

[1] plane sweep method
[2] sweep line
[3] scan
[4] event points
[5] priority queue

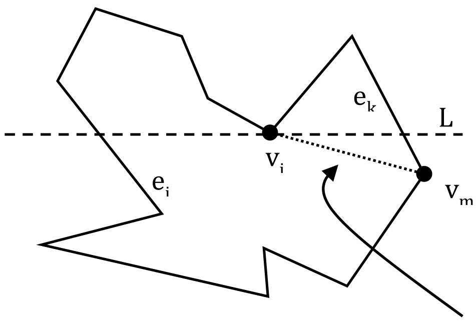
مولفه Y آنها مشخص می‌شود. هر نقطه‌ای که مقدار مولفه Y آن بزرگتر باشد اولویت بیشتری دارد. اگر دو راس مختلف دارای مولفه Y یکسان باشند، راسی اولویت بیشتری دارد که مقدار مولفه X آن کمتر است. با استفاده از این روش می‌توان راس پردازشی بعدی را در $O(\log n)$ یافت زیرا هزینه حذف یک عنصر از صف اولویت، از مرتبه $O(\log n)$ است. هدف از به کار بردن صف اولویت این است که راس‌های چندضلعی با توجه به مقدار مولفه Y آنها، از بالا به پایین ملاقات شوند. به جای استفاده از صف اولویت، می‌توان راس‌های چندضلعی را با توجه به مقدار مولفه Y آنها، در زمان $O(n \log n)$ مرتب کرد و سپس در زمان $O(1)$ راس ها را یکی پس از دیگری پردازش کرد.

هدف از پویش چندضلعی، اضافه کردن قطر از راس‌های ادغام به راس‌هایی است که در پایین آنها واقع شده‌اند (و همچنین اضافه کردن قطر از راس‌های تفکیک به راس‌هایی که در بالای آنها واقع شده‌اند). فرض شود خط جاروب به یک راس تفکیک مانند v_i می‌رسد. یک سوال مهم این است که راس v_i به چه راسی متصل شود؟ یک گزینه مناسب، راسی است که نزدیک v_i قرار دارد زیرا احتمالا می‌توان این دو راس را به یکدیگر متصل نمود بدون اینکه ضلعی از چندضلعی P و یا قطر دیگری از چندضلعی قطع شود. فرض شود e_j ضلعی است که (بلافاصله) در سمت چپ راس v_i قرار دارد (خط جاروب ضلع e_j را قطع می‌کند). این وضعیت در شکل ۲-۱۴ نمایش داده شده است. همواره می‌توان راس v_i را به پایین‌ترین راسی که بین ضلع‌های e_i و e_k قرار گرفته و بالاتر از v_i است متصل نمود. اگر چنین راسی وجود نداشته باشد، می‌توان v_i را به راس بالایی ضلع e_j و یا e_k متصل کرد. چنین راسی را نقطه کمکی e_j می‌نامیم و آن را با $helper(e_j)$ نشان می‌دهیم. نقطه کمکی e_j پایین‌ترین راسی است که بالای خط جاروب قرار دارد به طوری که پاره‌خط افقی که این راس را به ضلع e_j متصل می‌کند تماما درون چندضلعی P واقع می‌شود. باید توجه داشت که نقطه کمکی e_j می‌تواند راس بالایی e_j باشد.



شکل ۲-۱۴) پردازش یک راس تفکیک در الگوریتم *لی*

با استفاده از روشی که تشریح شد، می‌توان راس‌های تفکیک را از بین برد (آنها را به راس معمولی تبدیل کرد). در نگاه اول، از بین بردن راس‌های ادغام کاری سخت‌تر به نظر می‌رسد زیرا این راس‌ها باید به راس‌هایی که در پایین آنها قرار گرفته‌اند متصل شوند اما خط جاروب هنوز به آنها نرسیده است (هنوز پویش نشده‌اند). فرض کنید خط پویش به راس ادغامی مانند v_i می‌رسد. همچنین فرض کنید e_j و e_k اضلاعی باشند که در سمت چپ و راست v_i واقع شده‌اند. این وضعیت در شکل ۲-۱۵ نمایش داده شده است.



وقتی خط جاروب به راس v_m برسد، این قطر اضافه می‌شود.

شکل ۲-۱۵) پردازش یک راس ادغام

ملاحظه می‌شود که وقتی خط جاروب به این راس می‌رسد، این راس تبدیل به راس کمکی جدیدی برای ضلع e_j می‌شود. هدف این است که راس v_i به بالاترین راسی که در زیر خط جاروب و بین ضلع‌های e_j و e_k قرار دارد متصل شود. این هدف دقیقا عکس هدفی است که برای راس‌های تفکیک وجود دارد. علت این است که راس‌های تفکیک همان راس‌های ادغام هستند که وارونه شده‌اند. اگر یک چندضلعی ۱۸۰ درجه دوران داده شود، راس‌های تفکیک به راس ادغام و راس‌های ادغام به راس تفکیک تبدیل

می‌شوند. باید توجه داشت که وقتی خط جاروب به راس v_i می‌رسد، بالاترین راسی که زیر خط جاروب قرار دارد هنوز شناخته شده نیست اما بعدا می‌توان این راس را یافت. وقتی خط جاروب به راسی مانند v_m می‌رسد و این راس جایگزین راس v_i (به عنوان راس کمکی ضلع e_j) می‌شود، راس مورد نظر را یافته‌ایم. بنابراین هرگاه راس کمکی یک ضلع جایگزین می‌شود، این موضوع بررسی می‌شود که آیا راس قدیمی یک راس ادغام بوده است و یا خیر. اگر این چنین باشد، راس کمکی قدیمی با یک قطر به راس کمکی جدید متصل می‌شود. وقتی راس کمکی جدید یک راس تفکیک است، این قطر همواره به چندضلعی اضافه می‌شود (برای از بین بردن راس تفکیک). اگر راس قدیمی یک راس ادغام باشد، با یک قطر به طور همزمان یک راس ادغام و یک راس تفکیک از بین می‌رود. این احتمال وجود دارد که با پایین آمدن خط جاروب، نقطه کمکی ضلع e_j با راس دیگری جایگزین نشود. در این حالت راس v_i به راس پایینی ضلع e_j متصل می‌شود.

برای اجرای این الگوریتم، می‌بایست ضلع سمت راست هر راس شناسایی شود. ضلع‌هایی از P که خط جاروب را قطع می‌کنند در برگ‌های یک درخت دودویی[1] به نام T ذخیره می‌شوند. اضلاع چندضلعی P از سمت چپ به راست در برگ های T ذخیره می‌شوند. از آنجایی که تنها نیاز به ضلع‌های سمت چپ راس‌های ادغام و تفکیک وجود دارد، فقط ضلع‌هایی از P در درخت T نگهداری می‌شود که ناحیه داخلی چندضلعی P در سمت راست آنها واقع شده باشد. به همراه هر ضلع، راس کمکی آن نیز نگهداری می‌شود. درخت T و راس‌های کمکی که به همراه اضلاع چندضلعی در T ذخیره شده‌اند، وضعیت فعلی اجرای الگوریتم را نشان می‌دهد. با پایین آمدن خط جاروب، این وضعیت تغییر می‌کند. اضلاع مختلفی خط جاروب را در هنگام پایین آمدن قطع می‌کنند و راس کمکی یک یال ممکن است تغییر کند.

[1] binary tree

در ادامه، پیاده‌سازی این الگوریتم با استفاده از شش رویه[۱] ملاحظه می‌شود[۲]. رویه اصلی این الگوریتم MakeMonotone نام دارد که یک چندضلعی ساده را به عنوان ورودی دریافت می‌کند. پنج رویه دیگر راس‌های شروع، پایان، ادغام، تفکیک و معمولی را پردازش می‌کنند. رویه MakeMonotone راس‌های چندضلعی را یکی پس از دیگری ملاقات می‌کند و با توجه به نوع آن راس، رویه مربوط به آن را فراخوانی می‌کند.

Algorithm Make-Monotone(P)
Input: A simple polygon P stored in a doubly-connected edge list D.
Output: A partitioning of P into monotone subpolygons, stored in D.

1. Construct a priority queue Q on the vertices of P, using their y-coordinates as priority. If two points have the same y-coordinate, the one with smaller x-coordinate has higher priority.
2. Initialize an empty binary search tree T.
3. **while** Q is not empty
4. **do** Remove the vertex v_i with the highest priority from Q
5. Call the appropriate procedure to handle the vertex, depending on its type.

الگوریتم ۲-۱) الگوریتم تجزیه چندضلعی با الگوریتم لی و پریپاراتا

HandleStartVertex(v_i)

1. Insert e_i in T and set helper(e_i) to v_i.

ادامه الگوریتم ۲-۱) پردازش راس‌های شروع

HandleEndVertex(v_i)

1. **if** helper(e_{i-1}) is a merge vertex **then**
2. Insert the diagonal connecting v_i to helper (e_{i-1})
3. Delete e_{i-1} from T

ادامه الگوریتم ۲-۱) پردازش راس‌های پایان

---

[1] procedure

[۲] شبه کد مربوط به پیاده‌سازی این الگوریتم از مرجع [۱] اخذ شده است.

```
HandleSplitVertex(v_i)
    Search in T to find the edge e_j directly left of v_i.
    Insert the diagonal connecting v_i to helper (e_j) in D.
    helper(e_j) ← v_i
    Insert e_i in T and set helper(e_i) to v_i.
```

ادامه الگوریتم ۲-۱) پردازش راس‌های تفکیک

```
HandleMergeVertex(v_i)
    if helper(e_j-1) is a merge vertex
        then Insert the diagonal connectiong v_i to helper(e_i-1) in D
    Delete e_i-1 from T.
    Search in T to find the edge e_j directly left of v_i.
    if helper(e_j) is a merge vertex
        then Insert the diagonal connecting v_i to helper(e_j) in D.
    helper(e_j) ← v_i
```

ادامه الگوریتم ۲-۱) پردازش راس‌های ادغام

```
HandleRegularVertex(v_i)
    if the interior of P lies to the right of v_i
        then if helper(e_j-1) is a merge vertex
            then Insert the diagonal connecting v_i to helper(e_i-1) in D.
            Delete e_i-1 from T.
            Insert e_i in T and set helper(e_i) to v_i.
        else Search in T to find the edge ej directly left of v_i.
            if helper(e_j) is a merge vertex
              then Insert the diagonal connecting v_i to helper(e_j) in D.
            helper(e_j) ← v_i
```

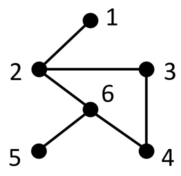
ادامه الگوریتم ۲-۱) پردازش راس‌های معمولی

۲-۳ الگوریتم لیو و نتافوس

الگوریتم لی و پریپاراتا کرانی روی تعداد چندضلعی‌های حاصل از تجزیه قرار نمی‌دهد. در مواردی مطلوب است عمل تجزیه به صورت کمینه انجام شود [۱۶]. لیو و نتافوس یک الگوریتم از مرتبه $O(nN^3 + N^2 n \log n + N^5)$ برای تجزیه کمینه یک چندضلعی ساده به چندضلعی‌های یکنواخت ارائه کرده‌اند [۵]. آنها برای تجزیه کمینه یک چندضلعی ساده از مسئله «مجموعه مستقل بیشینه» کمک گرفتند. این مسئله در حوزه مسائل نظریه گراف است و تا کنون الگوریتم‌های متعددی برای حل آن ارائه شده است. لیو و نتافوس نیز در کنار الگوریتم پیشنهادی خود برای تجزیه کمینه چندضلعی، یک الگوریتم برای حل این مسئله ارائه کردند.

مجموعه مستقل بیشینه یک گراف مدور

یکی از گام‌های اساسی الگوریتم لیو و نتافوس برای تجزیه کمینه چندضلعی، یافتن یک مجموعه مستقل بیشینه یک گراف مدور[1] است. یک مجموعه مستقل[2] در یک گراف بدون جهت، زیرمجموعه‌ای از راس-های آن است به طوری که هیچ دو راسی (از این مجموعه) توسط یک یال به هم متصل نشده باشد. به بزرگترین مجموعه مستقل یک گراف، مجموعه مستقل بیشینه[3] گفته می‌شود. مجموعه مستقل بیشینه یک گراف الزاما یکتا نیست.



$\mathbf{S_1} = \{1, 4\}$
$\mathbf{S_2} = \{1, 3, 6\}$
$\mathbf{S_3} = \{1, 4, 5\}$
$\mathbf{S_4} = \{4, 5\}$

شکل ۲-۱۶) یک گراف و چهار مجموعه مستقل آن

[1] circle graph
[2] independent set
[3] maximum independent set

شکل ۲-۱۶ یک گراف و چهار مجموعه مستقل آن را نشان می‌دهد. مجموعه‌های S_2 و S_3 دو زیرمجموعه مستقل بیشینه این گراف هستند. این گراف مجموعه‌های مستقل دیگری نیز دارد. مسئله یافتن مجموعه مستقل بیشینه یک گراف بدون جهت، یک مسئله NP-سخت [1] است [۲۴]. با این حال الگوریتم‌های با پیچیدگی زمانی چندجمله‌ای برای یافتن مجموعه مستقل بیشینه یک گراف مدور وجود دارد [۲۳،۲۲،۲۱،۲۰،۱۹،۱۸،۱۷،۵].

گراف تقاطع[2] قطرهای یک دایره مفروض به نام C، گراف مدور[3] متناظر با دایره C نامیده می‌شود. بر خلاف تعریف رایج قطر، نیازی نیست که قطرها حتما از مرکز دایره بگذرند. گراف مدور متناظر با دایره مفروض C به این صورت تشکیل می‌شود؛ به ازای هر قطر دایره C یک راس در گراف مدور در نظر گرفته می‌شود. اگر دو قطر در دایره C یکدیگر را قطع کنند، دو راس متناظر با آن (در گراف مدور) با یک یال به هم متصل می‌شوند. باید توجه داشت که هر گرافی، گراف مدور نیست [۲۴]. تا به حال الگوریتم‌های متعددی برای یافتن مجموعه مستقل بیشینه یک گراف مدور ارائه شده است [۱۵،۱۴،۱۳،۱۲،۱۱،۱۰،۹،۴]. *لیو* و *نتافوس* در کنار الگوریتم پیشنهادی خود برای تجزیه کمینه چندضلعی، یک الگوریتم از مرتبه $O(n^3)$ برای یافتن یک مجموعه مستقل بیشینه یک گراف مدور با n راس ارائه کردند [۴].

الگوریتم لیو و نتافوس برای تجزیه کمینه یک چندضلعی

لیو و *نتافوس* برای تجزیه کمینه یک چندضلعی ساده از اصولی تقریبا مشابه با اصول الگوریتم ارائه شده توسط *لی* و *پریپاراتا* استفاده کردند. آنها برای تجزیه یک چندضلعی پیشنهاد دادند که راس‌های ادغام توسط قطرهایی به راس‌هایی که پایین‌تر از آنها قرار دارند و قابل دید[4] است متصل شود. همچنین برای

[1] NP-Hard
[2] intersection graph
[3] circle graph
[4] Visible

از بین بردن راس‌های تفکیک پیشنهاد دادند که این راس‌ها توسط قطرهایی به راس‌هایی که بالاتر از آنها قرار دارد و قابل دید است متصل شود. از آنجایی که با اضافه شدن هر قطر به چندضلعی تعداد چندضلعی‌های حاصل از فرآیند تجزیه یک واحد افزایش پیدا می‌کند، یک تجزیه کمینه زمانی حاصل می‌شود که کمترین تعداد قطرها به چندضلعی اضافه شود. با توجه به اینکه باید به ازای هر راس ادغام یا تفکیک یک قطر اضافه شود (تا آن راس به یک راس معمولی تبدیل شود)، *لیو* و *نتافوس* پیشنهاد دادند تا جایی که امکان دارد، با هر قطر یک راس ادغام و یک راس تفکیک به یکدیگر متصل شود.
با استفاده از این روش، با هر قطر دو راس چرخشی[1] از بین می‌رود. البته باید توجه داشت که راس ادغام می‌بایست بالاتر از راس تفکیک واقع شده باشد. با محاسبه مجموعه بیشینه قطرهایی که یک راس ادغام و یک راس تفکیک را به هم متصل می‌کنند، می‌توان یک چندضلعی ساده را به صورت کمینه تجزیه کرد. الگوریتم *لیو* و *نتافوس* در چهار گام اجرا می‌شود:

گام اول: همه راس‌های ادغام و تفکیک چندضلعی یافته می‌شود. به ازای هر راس ادغام یا تفکیک یک راس روی محیط دایره‌ای به نام C قرار می‌گیرد. راس‌ها به همان ترتیبی روی محیط C قرار می‌گیرند که در چندضلعی قرار گرفته‌اند (ترتیب راس‌ها حفظ می‌شود).

گام دوم: در چندضلعی، همه زوج راس‌هایی که شامل یک راس ادغام و یک راس تفکیک می‌باشند و می‌توان آن‌ها را با قطر به یکدیگر متصل کرد (یکدیگر را می‌بینند[2]) یافته می‌شود. به ازای هر یک از این زوج راس‌ها، راس‌های متناظر با آنها در دایره C با یک قطر به یکدیگر متصل می‌شوند.

گام سوم: مجموعه مستقل بیشینه گراف مدور متناظر با دایره C یافته می‌شود. هر راس از این مجموعه یک قطر را نشان می‌دهد. به ازای هر راس از این مجموعه قطر متناظر با آن راس به چندضلعی P اضافه می‌شود.

[1] turn vertex
[2] visible to each other

گام چهارم: راس‌های ادغام و تفکیکی که توسط هیچ قطری به راس‌های دیگر متصل نشده‌اند، با استفاده از الگوریتم لی و پریپاراتا به راس مناسب متصل می‌شوند [۲۵].

تحلیل پیچیدگی زمانی

اجرای گام اول مستلزم پیمایش راس‌های چندضلعی P است. زمان اجرای این کار از مرتبه $O(n)$ است. در گام دوم، یک الگوریتم از مرتبه $O(n)$ برای یافتن چندضلعی آشکار[1] از یک راس بکار برده می‌شود. چندضلعی آشکار از یک نقطه مانند x، مجموعه نقاطی از چندضلعی P است که از x آشکار است. در این گام $O(N)$ چندضلعی آشکار محاسبه می‌شود. با پیمایش هر کدام از این چندضلعی‌های آشکار زوج راس‌های ادغام و تفکیکی که می‌توان آنها را با یک قطر به هم متصل کرد، یافته می‌شود. بنابراین زمان اجرای این گام از مرتبه $O(nN)$ است. زمان اجرای گام سوم از مرتبه $O(N^3)$ است [۵]. گام چهارم در زمان $O(n \log n)$ توسط الگوریتم تجزیه یکنواخت *لی* و *پریپاراتا* اجرا می‌شود [۲۵]. بنابراین این الگوریتم یک چندضلعی ساده را در زمان $O(n \log n + nN + N^3)$ به صورت کمینه به چندضلعی‌های یکنواخت قائم تجزیه می‌کند.

با وجود اینکه این تجزیه نسبت به محور Y کمینه است اما این امکان وجود دارد که در مقایسه با بقیه خطوط کمینه نباشد. به عبارت دیگر امکان دارد خطی مانند W وجود داشته باشد که تجزیه کمینه یکنواخت نسبت به W منجر به تولید چندضلعی‌های کمتری در مقایسه با تجزیه کمینه یکنواخت نسبت به محور Y شود. برای یافتن خطی که تجزیه کمینه یکنواخت نسبت به آن، کمترین تعداد چندضلعی را به نسبت بقیه خطوط تولید می‌کند، نیاز به اجرای الگوریتم فوق به ازای $O(N^2)$ خط است [۵]. بنابراین الگوریتم *لیو* و *نتافوس* یک چندضلعی ساده را در زمان $O(nN^2 \log n + nN^3 + N^5)$ به صورت کمینه (نسبت به هر خطی) به چندضلعی‌های یکنواخت تجزیه می‌کند.

[1] visibility polygon

۲-۴ الگوریتم مثلثی‌سازی سایدل

مثلثی‌سازی یک چندضلعی یکی از مهمترین و پرکاربردترین مسائل در حوزه هندسه محاسباتی است. طی چندین دهه الگوریتم‌های متعددی برای حل این مسئله ارائه شده است. تا به حال الگوریتم‌های غیر خطی زیادی برای حل این مسئله ارائه شده است. برای سال‌های متمادی، این پرسش که آیا یک الگوریتم از مرتبه $O(n)$ برای مثلثی‌سازی یک چندضلعی ساده با n راس وجود دارد یا خیر بدون پاسخ باقی مانده بود. در نهایت در سال ۱۹۹۱ *چازل* یک الگوریتم خطی (کاملا پیچیده) برای حل این مسئله ارائه کرد [۲۶]. پیچیدگی پیاده‌سازی این الگوریتم باعث شده است که در عمل از الگوریتم‌های تقریبا خطی استفاده شود[1]. بعد از ارائه این الگوریتم توسط *چازل*، تلاش برای یافتن الگوریتم‌های ساده‌تر متوقف نشد. در سال ۱۹۹۴ *سایدل*[2] یک الگوریتم تصادفی افزایشی[3] برای مثلثی‌سازی یک چندضلعی ارائه کرد [۷]. این الگوریتم در سه گام اجرا می‌شود:

گام اول: ابتدا چندضلعی به صورت ذوزنقه‌ای تجزیه می‌شود. برای این منظور سایدل یک الگوریتم تصادفی افزایشی با پیچیدگی زمانی مورد انتظار $O(n \log^* n)$ و میزان حافظه مصرفی مورد انتظار $O(n)$ ارائه کرد (n تعداد راس‌های چندضلعی است). نماد $\log^* n$ به صورت بازگشتی تعریف می‌شود. با فرض اینکه $\log^{(i)} n = \log(\log^{(i-1)} n)$ و همچنین $\log^{(0)} n = n$ است، نماد $\log^* n$ بزرگترین عدد صحیح مانند L را نشان می‌دهد به طوری که $\log^{(L)} n \geq 1$.

گام دوم: چندضلعی با کمک تجزیه ذوزنقه‌ای به صورت یکنواخت قائم تجزیه می‌شود. این کار در زمان $O(n)$ انجام می‌شود.

[1] تا زمان نگارش این پایان‌نامه، هیچ پیاده‌سازی از الگوریتم مثلثی‌سازی آقای چازل سراغ نداریم. به دلیل اینکه پیاده‌سازی این الگوریتم پیچیده است، در عمل از الگوریتم‌های تقریبا خطی مانند الگوریتم سایدل استفاده می‌شود.

[2] Seidel

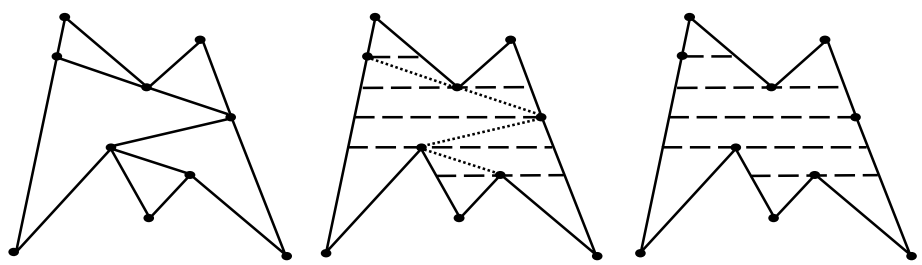
[3] incremental randomized algorithm

گام سوم: هر کدام از چندضلعی‌های یکنواخت قائم حاصل از تجزیه، در زمان $O(n)$ مثلثی‌سازی می‌شوند. برای مثلثی‌سازی یک چندضلعی یکنواخت قائم در زمان خطی، الگوریتم‌های بسیار ساده‌ای وجود دارد [۱] و نیازی به استفاده از یک الگوریتم خطی پیچیده مانند الگوریتم *چازل* نیست.

زمان اجرای مورد انتظار این الگوریتم از مرتبه $O(n \log^* n)$ و میزان حافظه مصرفی مورد انتظار آن از مرتبه $O(n)$ است. با توجه به اینکه در گام دوم الگوریتم چندضلعی به صورت یکنواخت قائم تجزیه می‌شود، می‌توان از الگوریتم سایدل برای تجزیه یکنواخت یک چندضلعی استفاده کرد (گام سوم اجرا نمی‌شود). با توجه به اینکه $\log^* n$ به مراتب کوچکتر از $\log n$ است، در حالت کلی انتظار داریم که الگوریتم سایدل سریع‌تر از الگوریتم لی و پریپاراتا عمل کند. الگوریتم سایدل نیز مانند الگوریتم لی و پریپاراتا هیچ کرانی را روی تعداد چندضلعی‌های یکنواخت حاصل از تجزیه قرار نمی‌دهد.

۲-۴-۱ استفاده از الگوریتم سایدل برای تجزیه یکنواخت یک چندضلعی

همانطور که بیان شد، می‌توان از الگوریتم مثلثی‌سازی سایدل برای تجزیه یکنواخت یک چندضلعی استفاده کرد. به دلیل اینکه جزئیات گام اول این الگوریتم خارج از حوزه این تحقیق است، به آنها نمی‌پردازیم. بنابراین فرض می‌کنیم که چندضلعی به صورت ذوزنقه‌ای تجزیه شده است. در شکل ۲-۱۷ (الف) یک چندضلعی که به صورت افقی (تجزیه) ذوزنقه‌ای شده است ملاحظه می‌شود. این چندضلعی به ۹ ذوزنقه تجزیه شده است.



(ج) **(ب)** **(الف)**

شکل ۲-۱۷) مراحل تجزیه یکنواخت یک چندضلعی ساده با استفاده از الگوریتم سایدل

در گام دوم الگوریتم سایدل، به ازای هر ذوزنقه بررسی می‌شود آیا دو راسی که روی قاعده بالایی و پایینی ذوزنقه قرار گرفته‌اند با یک ضلع (از اضلاع چندضلعی) به یکدیگر متصل شده‌اند یا خیر. اگر این دو راس با یک ضلع به یکدیگر متصل نباشند، توسط یک قطر به یکدیگر متصل می‌شوند. اگر این دو راس توسط یک ضلع به یکدیگر متصل شده باشند، عملی روی ذوزنقه صورت نمی‌گیرد و ذوزنقه بعدی بررسی می‌شود. قطرهایی که به این صورت به چندضلعی اضافه می‌شوند، آن را به مجموعه‌ای از چندضلعی‌های یکنواخت قائم تجزیه می‌کنند. چندضلعی تجزیه شده نهایی در شکل ۲-۱۷ (ج) ملاحظه می‌شود.

۲-۴ جمع‌بندی و نتیجه‌گیری

در این فصل چندین الگوریتم برای تجزیه یک چندضلعی به مجموعه‌ای از چندضلعی‌های یکنواخت مورد بررسی قرار گرفت. معمولا برای مقایسه کارآیی و کیفیت خروجی این الگوریتم‌ها از چهار شاخص استفاده می‌شود. این شاخص‌ها عبارتند از:

۱. پیچیدگی زمان و فضای الگوریتم
۲. تعداد چندضلعی‌های حاصل از تجزیه
۳. ایجاد و یا عدم ایجاد نقاط کمکی[۱]
۴. توانایی تجزیه چندضلعی‌های حفره‌دار

شاخص اول برای ارزیابی کارآیی هر الگوریتمی مورد بررسی قرار می‌گیرد. در بسیاری از کاربردهای هندسه محاسباتی مانند گرافیک برداری[۲] و چندضلعی‌های مورد استفاده در سامانه‌های اطلاعات جغرافیایی[۳] تعداد راس‌های چندضلعی و یا تعداد کل چندضلعی‌ها به چندین هزار می‌رسد [۸]. پردازش

---

[۱] steiner points
[۲] vector graphic
[۳] Geographical Information Systems (GIS)

این چندضلعی‌ها در یک زمان قابل قبول، مخصوصا در کاربردهای بلادرنگ[۱] مانند بازی‌های رایانه‌ای که هر صحنه باید به سرعت ارائه[۲] و به کاربر نمایش داده شود، می‌تواند چالش برانگیز باشد.

تعداد چندضلعی‌های حاصل از تجزیه، یکی دیگر از معیارهای بررسی این دسته از الگوریتم‌ها است. در کاربردهایی مانند بینایی ماشین[۳] و نویسه خوانی نوری[۴] مطلوب‌تر این است که تعداد چندضلعی‌های حاصل از تجزیه، کمینه و یا تقریبا کمینه باشد.

به هر اندازه که تعداد چندضلعی‌های حاصل از تجزیه کمتر شود، تعداد راس‌های هر چندضلعی افزایش پیدا کرده و چندضلعی‌های بزرگتری حاصل می‌شود. از نظر بصری[۵] چندضلعی‌های بزرگتر نتایج بهتری را فراهم می‌کنند. البته در برخی از کاربردهای تجزیه چندضلعی مانند مثلثی‌سازی، تعداد چندضلعی‌های یکنواخت حاصل از تجزیه تاثیری روی کیفیت نهایی نتایج و یا زمان اجرای الگوریتم ندارد.

شاخص دیگری که برای مقایسه این الگوریتم‌ها وجود دارد این است که آیا بعد از تجزیه یک چندضلعی تعداد راس‌های آن افزایش پیدا می‌کند یا خیر. به این راس‌های اضافی که بعد از تجزیه چندضلعی پدید می‌آیند، نقاط کمکی گفته می‌شود. این گونه الگوریتم‌ها از این نقاط برای تجزیه چندضلعی کمک می‌گیرند. هرچه تعداد این نقاط بیشتر باشد، پردازش‌های بعدی که می‌بایست روی چندضلعی تجزیه شده انجام شود با سرعت کمتری اجرا می‌شوند. استفاده نکردن از نقاط کمکی یک مزیت برای الگوریتم تجزیه محسوب می‌شود.

شاخص دیگر برای مقایسه الگوریتم‌های تجزیه، امکان اجرای الگوریتم روی چندضلعی‌های حفره‌دار است. برخی از الگوریتم‌ها مانند الگوریتم *لی* فقط روی چندضلعی‌های ساده قابل اجرا هستند. اینکه یک

[1] real time
[۲] render
[۳] machine vision
[۴] Optical Character Recognition (OCR)
[۵] visual

الگوریتم قادر باشد چندضلعی‌های حفره‌دار را نیز تجزیه کند، یک مزیت محسوب می‌شود زیرا در عمل با چنین چندضلعی‌هایی مواجه می‌شویم.

در جدول شماره ۲-۱ الگوریتم‌های شناخته شده‌ای که برخی از آنها در این فصل مورد مطالعه قرار گرفت، با استفاده از چهار شاخصی که بیان شد با یکدیگر مقایسه شده‌اند. در این جدول، n تعداد راس‌های چندضلعی، h تعداد حفره‌های چندضلعی و N تعداد راس‌هایی که زاویه داخلی آنها بیشتر از ۱۸۰ درجه است را نشان می‌دهد.

جدول ۲-۱) برخی از الگوریتم‌های شناخته شده برای تجزیه یکنواخت یک چندضلعی

طراح الگوریتم	پیچیدگی زمان اجرا و حافظه	تجزیه کمینه	استفاده از نقاط کمکی	توانایی تجزیه چندضلعی حفره‌دار
لی و پریپاراتا [۱]	$T(n) \in O(n \log n)$ $S(n) \in O(n)$	خیر	خیر	خیر
لیو و نتافوس [۵]	$T(n) \in O(nN^3 + N^2 n \log n + N^5)$ $S(n) \in O(n^2)$	بله	خیر	خیر
لیو و نتافوس [۵]	$T(n) \in O(n\, N^3 \log n + N^5)$ $S(n) \in O(n^2)$	بله	بله	خیر
سایدل [۷]	ERT[۱] $\in O(n \log^* n)$ EMU[2] $\in O(n)$	خیر	خیر	بله[۳]
وی [۲۷]	$T(n) \in O(K(n \log n + h \log^3 h))$ $S(n) \in O(n)$	بله	بله	بله
کیل [۴]	$T(n) \in O(n^4 N)$	بله[۴]	خیر	خیر

زمان اجرای الگوریتم *لی* برای بعضی از کاربردهای عملی قابل قبول است اما این الگوریتم هیچ تلاشی برای کمینه کردن تعداد چندضلعی‌های حاصل از تجزیه انجام نمی‌دهد. همچنین این الگوریتم قادر به تجزیه چندضلعی‌های حفره‌دار نیست. از طرف دیگر، الگوریتم *لیو* و *نتافوس* عمل تجزیه را به

[۱] Expected Running Time (ERT)

[۲] Expected Memory Usage (EMU)

[۳] در مقاله *سایدل* به این موضوع اشاره‌ای نشده است اما با اندکی ویرایش این الگوریتم می‌توان چندضلعی‌های حفره‌دار را نیز تجزیه کرد [۲۲].

[۴] تجزیه به صورت کمینه انجام می‌گیرد اما الزاما یکپارچه یکنواخت (uniformly monotone) نیست [۴].

صورت کمینه انجام می‌دهد اما زمان اجرای آن مطلوب نیست. با توجه اینکه در بدترین حالت N از مرتبه $O(n)$ می‌باشد[۱]، زمان اجرای بدترین حالت این الگوریتم از مرتبه $O(n^5)$ است.

از نظر زمان اجرا، الگوریتم *سایدل* یک الگوریتم کارا برای تجزیه یکنواخت یک چندضلعی است. زمان اجرای مورد انتظار $O(n \log^* n)$ یک زمان اجرای تقریبا خطی محسوب می‌شود. با وجود اینکه این الگوریتم زمان اجرای مطلوبی دارد اما چندضلعی‌ها را به تعداد زیادی چندضلعی یکنواخت تجزیه می‌کند. این الگوریتم نه تنها تلاشی برای کنترل تعداد چندضلعی‌های حاصل از تجزیه نمی‌کند بلکه در اکثر موارد قطرهایی به چندضلعی اضافه می‌کند که نیازی به آنها نیست. البته با توجه به اینکه این الگوریتم اساسا برای مثلثی‌سازی یک چندضلعی ارائه شده است، این رفتار الگوریتم تاثیری روی زمان اجرا و یا کیفیت مثلثی‌سازی ندارد (یک چندضلعی ساده با n راس به هر صورت که مثلثی‌سازی شود در نهایت به n-2 مثلث تجزیه می‌شود). از مقایسه این الگوریتم‌ها با یکدیگر می‌توان به این نتیجه رسید که رابطه معکوسی بین زمان اجرا و تجزیه مطلوب‌تر (کمینه بودن تجزیه، استفاده نکردن از نقاط کمکی و توانایی تجزیه چندضلعی‌های حفره‌دار) وجود دارد. در سال ۱۹۸۳ *کیل* ثابت کرد که مسئله تجزیه کمینه یک چندضلعی حفره‌دار به چندضلعی‌های یکنواخت، بدون استفاده از نقاط کمکی، یک مسئله NP-سخت است [۴].

---

[۱] مقدار N وابسته به n است. در حالت کلی، هرچه مقدار n بیشتر می‌شود، مقدار N نیز به صورت خطی افزایش پیدا می‌کند.

فصل ۳

ارائه یک الگوریتم حریصانه جهت تجزیه یکنواخت چند ضلعی

در این فصل یک الگوریتم حریصانه[1] جهت تجزیه یکنواخت یک چندضلعی حفره‌دار، بدون استفاده از نقاط کمکی، ارائه می‌شود. 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. در بخش ۳-۱ ابتدا ایده اصلی این الگوریتم در قالب یک الگوریتم اولیه مطرح می‌شود، سپس در بخش ۳-۲ زمان اجرای الگوریتم اولیه با استفاده از روش‌هایی بهبود داده می‌شود. در بخش ۳-۳ نیز کارآیی الگوریتم جدید با الگوریتم *لی* و الگوریتم *سایدل* مقایسه می‌شود.

۳-۱ الگوریتم اولیه

در این بخش یک الگوریتم مقدماتی که به صورت حریصانه یک چندضلعی ساده را به چندضلعی‌های یکنواخت تجزیه می‌کند ارائه می‌شود. این الگوریتم از مفهوم «چندضلعی آشکار» برای تجزیه چندضلعی استفاده می‌کند. بعد از تشریح نحوه عملکرد الگوریتم، پیچیدگی فضا و زمان آن تحلیل می‌شود.

[1] greedy algorithm

۳-۱-۲ الگوریتم اولیه

قضیه چندضلعی یکنواخت قائم بیان می‌کند برای اینکه یک چندضلعی ساده یکنواخت قائم باشد، نباید راس‌های ادغام یا تفکیک داشته باشد. هدف تمام الگوریتم‌هایی که برای تجزیه یکنواخت چندضلعی‌ها ارائه شده است این است که با اضافه کردن قطرهایی به چندضلعی، این راس‌ها را از بین ببرند.

الگوریتمی که در اینجا ارائه می‌کنیم از این اصل کلی مستثنی نیست. این الگوریتم برای یافتن قطرها از چندضلعی آشکار استفاده می‌کند. همچنین برای کاهش تعداد چندضلعی‌های حاصل از تجزیه سعی می‌شود تا جایی که امکان دارد (به صورت حریصانه) راس‌های ادغام و تفکیک به یکدیگر متصل شوند. برای اینکه قطرهایی که یک انتهای آن به راس ادغام و انتهای دیگر آن به راس تفکیک متصل است از بقیه قطرها متمایز شود، تمامی قطرها را در سه دسته طبقه‌بندی می‌کنیم:

قطر ویژه: قطری که یک انتهای آن به راس ادغام و انتهای دیگر آن به راس تفکیک متصل است.

قطر معمولی: قطری که تنها یک انتهای آن به راس تفکیک و یا ادغام متصل است.

قطر غیر مفید: قطری که هر دو انتهای آن به راس‌هایی متصل است که نه ادغام هستند و نه تفکیک.

استفاده از قطرهای ویژه نسبت به قطرهای معمولی ارجحیت دارد زیرا این قطرها یک راس ادغام و یک راس تفکیک را به طور همزمان از بین می‌برند. هرچه از تعداد قطر ویژه بیشتری استفاده شود تعداد چندضلعی‌های حاصل از تجزیه کمتر می‌شود. الگوریتمی که در اینجا ارائه می‌کنیم، به صورت حریصانه سعی می‌کند تا جایی که امکان دارد راس‌های ادغام و تفکیک را با استفاده از قطرهای ویژه به هم متصل کند. بر خلاف الگوریتم لیو و نتافوس که قبل از استفاده از قطرهای ویژه، ابتدا «مجموعه بیشینه قطرهای ویژه» را محاسبه می‌کند، الگوریتمی که ارائه می‌دهیم به دو دلیل این مجموعه را محاسبه نمی‌کند. دلیل اول برای محاسبه نکردن این مجموعه، کاهش زمان اجرای الگوریتم است. دلیل دوم این است که در موارد بسیاری، انتخاب حریصانه قطرها منجر به جواب تقریبا بهینه و یا حتی بهینه می‌شود. در کاربردهایی که الزاما به تجزیه کمینه نیاز ندارند و تجزیه تقریبا کمینه نیز

نیازهای مسئله را برآورده می‌کند، انتخاب قطرهای ویژه به صورت حریصانه، در اغلب اوقات منجر به جواب‌های مطلوبی می‌شود.

این الگوریتم با شروع از راس‌های ادغام سعی می‌کند هر راس ادغام را به یک راس تفکیک متصل کند. اگر راس تفکیکی یافت نشود، راس ادغام به یک راس معمولی متصل می‌شود. بعد از بین بردن همه راس‌های ادغام، راس‌های تفکیکی که قطری به آنها متصل نشده است توسط یک قطر معمولی به نزدیکترین راس آشکار که بالای آنها واقع شده است متصل می‌شوند. شبه‌کد این روش در الگوریتم ۳-۱ ملاحظه می‌شود. در این الگوریتم از برخی توابع (کمکی) استفاده شده است که عبارتند از:

VIS(P, a): مجموعه راس‌های چندضلعی آشکار از نقطه a (درون چندضلعی P) را محاسبه می‌کند.
V(P): مجموعه راس‌های چندضلعی P را بر می‌گرداند.
Y(a): مقدار مولفه Y راس a را بر می‌گرداند.

Algorithm ElementryMonotoneDecomposition
Input: A simple polygon *P* with *n* vertices
Output: A set of diagonals that decompose *P* into Y-monotone polygons
Step 1: Find all split and merge vertices of *P* and put merge vertices in array *m* and split vertices in array *s.*
Step 2: // *Process merge vertices*
2.1 **for** each vertex *v* in *m* **do**
2.2 **if** there exist a split vertex *z* in VIS (*P*, *v*) ∩ V(*P*) such that Y(*z*) < Y(*v*) **then**
Add a diagonal from v to nearest *z*
else
Add a diagonal from *v* to nearest vertex in VIS (*P*, *v*) ∩ V (*P*)
Step 3: // *Process split vertices*
3.1 **for** each vertex *v* in *s* **do**
3.2 **if** *v* is not an endpoint of a diagonal **then**
3.3 Add a diagonal from *v* to nearest vertex in VIS (*P*, *v*) ∩ V(*P*)
End.

الگوریتم ۳-۱) الگوریتم حریصانه اولیه

تحلیل پیچیدگی زمانی الگوریتم ۳-۱

زمان اجرای گام اول از مرتبه $O(n)$ است. با یک بار پیمایش راس‌های چندضلعی می‌توان زاویه داخلی هر راس را محاسبه کرد. با استفاده از زاویه داخلی هر راس و مختصات راس‌های مجاور آن می‌توان نوع هر راس را مشخص کرد (راس ادغام، تفکیک، شروع، پایان و یا معمولی). زمان اجرای گام دوم به تعداد راس‌های ادغام بستگی دارد. فرض کنید تعداد راس‌های ادغام چندضلعی P برابر با N_m است.

در گام ۲-۲ یک چندضلعی آشکار محاسبه می‌شود. محاسبه هر چندضلعی آشکار به زمان $O(n)$ نیاز دارد. یافتن یک راس تفکیک در میان راس‌های چندضلعی آشکار محاسبه شده در گام ۲-۲ به زمان $O(n)$ نیاز دارد. بنابراین زمان اجرای گام دوم برابر است با:

$$t_2(n, N_m) = N_m\,(O(n) + O(n)) \in O(n \times N_m) \qquad (۱-۳)$$

در گام سوم به ازای هر راس تفکیک بررسی می‌شود که آیا این راس توسط یک قطر به راس ادغامی متصل شده است و یا خیر. اگر متصل نشده باشد، راس تفکیک به نزدیکترین راس آشکاری که بالای آن قرار دارد متصل می‌شود (چنین راسی حتما وجود دارد). در عمل گام سوم سریع‌تر از گام دوم اجرا می‌شود به این دلیل که برخی از راس‌های تفکیک نیازی به پردازش ندارند (زیرا در گام دوم با یک قطر به راس‌های ادغام متصل شده‌اند). فرض کنید تعداد راس‌های تفکیک چندضلعی P برابر N_s باشد. بنابراین زمان اجرای بدترین حالت گام سوم برابر است با:

$$t_3(n, N_s) = N_s \times O(n) \in O(n \times N_s) \qquad (۲-۳)$$

در نتیجه زمان اجرای الگوریتم ۳-۱ برابر مقدار زیر است:

$$t(n, N_m, N_s) = O(n) + O(n \times N_m) + O(n \times N_s) \in O(n) + O(n\,(N_m + N_s)) \qquad (۳-۳)$$

اگر مجموع تعداد راس‌های تفکیک و ادغام با N نمایش داده شود، در اینصورت:

$$t(n) \in O(n) + O(n \times N) \in O(n + n \times N) \qquad (۴-۳)$$

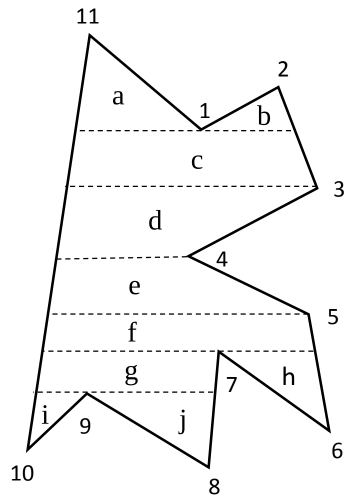
□

۳-۲ بهبود الگوریتم اولیه

زمان اجرای الگوریتم ۳-۱ از مرتبه $O(n+nN)$ است. بخش عمده‌ای از زمان اجرای این الگوریتم صرف محاسبه چندضلعی‌های آشکار می‌شود. در بدترین حالت به ازای هر راس ادغام یا تفکیک، یک چندضلعی آشکار محاسبه می‌شود. در این بخش الگوریتم بهبود یافته‌ای بر مبنای الگوریتم ۳-۱ ارائه می‌شود که بدون نیاز به محاسبه چندضلعی آشکار، عمل تجزیه را انجام می‌دهد. این الگوریتم همانند الگوریتم *سایدل*، از تجزیه ذوزنقه‌ای برای انجام تجزیه یکنواخت استفاده می‌کند اما تعداد چندضلعی‌های کمتری تولید می‌کند.

۳-۲-۱ حذف راس‌های ادغام و تفکیک به کمک تجزیه ذوزنقه‌ای

در اینجا به ارائه روشی پرداخته می‌شود که با استفاده از تجزیه ذوزنقه‌ای یک چندضلعی، سعی می‌کند یک راس ادغام و یک راس تفکیک را به یکدیگر متصل کند. برای تشریح بهتر عملکرد این روش، مراحل اجرای آن روی یک چندضلعی ساده را دنبال می‌کنیم. در شکل ۳-۱ یک چندضلعی که به صورت ذوزنقه‌ای تجزیه شده است ملاحظه می‌شود. این چندضلعی به ۱۰ ذوزنقه تجزیه شده است (مثلث‌ها نیز ذوزنقه‌هایی فرض شده‌اند که طول یکی از قاعده‌های آنها صفر است).



شکل ۳-۱) یک چندضلعی ساده که به صورت ذوزنقه‌ای تجزیه شده است.

ذوزنقه‌هایی که زیر یک راس ادغام قرار می‌گیرند (مانند ذوزنقه c) دارای دو همسایه بالایی هستند (ذوزنقه‌های a و b). به همین صورت ذوزنقه‌هایی که بالای یک راس تفکیک قرار گرفته‌اند (مانند ذوزنقه f) دارای دو همسایه پایینی هستند (ذوزنقه‌های g و h). فرض کنید تجزیه ذوزنقه‌ای به گونه‌ای انجام شده است که به ازای هر ذوزنقه، گره‌ای[1] در حافظه وجود دارد که حاوی اطلاعات زیر است:

۱. مختصات هندسی چهار راس ذوزنقه

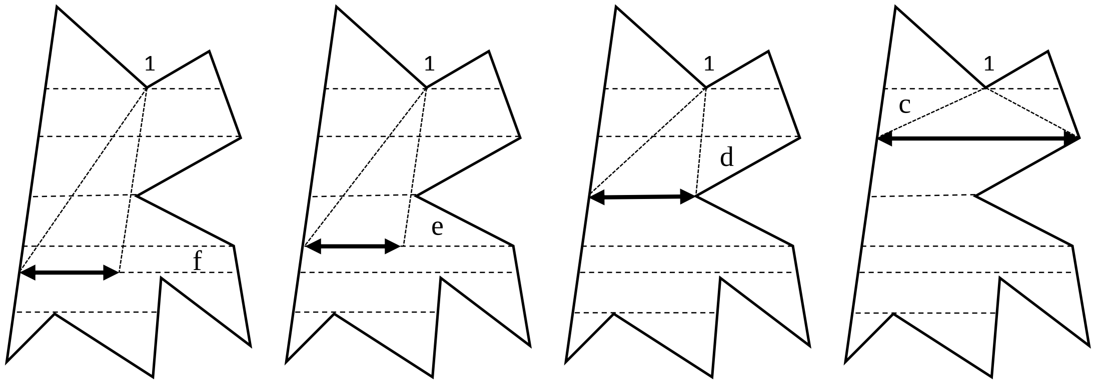
۲. اشاره‌گرهایی به ذوزنقه‌های بالایی و پایینی

مجموعه این گره‌ها یک درخت بدون ریشه را تشکیل می‌دهند. با استفاده از این اطلاعات می‌توان با شروع از یک ذوزنقه (گره) دلخواه، مجموعه ذوزنقه‌ها را به سمت بالا و یا پایین پیمایش کرد. همچنین فرض شود برای هر راس ادغام اشاره‌گری وجود دارد که به ذوزنقه‌ای که در زیر آن راس قرار دارد اشاره می‌کند و به همین ترتیب، برای هر راس تفکیک اشاره‌گری وجود دارد که به ذوزنقه‌ای که در بالای آن راس قرار گرفته است اشاره می‌کند.

می‌توان از این ساختمان داده برای اضافه کردن یک قطر ویژه به چندضلعی کمک گرفت. برای مثال 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. در هنگام پیمایش ذوزنقه‌ها، هرگاه با راس تفکیکی (مجاور با قاعده پایینی یک ذوزنقه) مواجه شویم که می‌توان آن را به راس شماره ۱ متصل کرد، این کار را انجام داده و پیمایش را متوقف می‌کنیم. در هنگام پیمایش ذوزنقه‌ها به سمت پایین، باید ملاحظاتی را در نظر داشته باشیم. باید به این نکته توجه داشت که ممکن است فقط بخشی از قاعده پایینی ذوزنقه‌ای که ملاقات می‌کنیم، از راس ادغام آشکار[2] باشد. در هنگام ملاقات هر ذوزنقه، بخشی از قاعده پایینی آن

[1] node
[2] visible

که از راس ادغام، آشکار[1] است محاسبه می‌شود. شکل ۳-۲ ناحیه آشکار از راس شماره ۱روی قاعده پایینی ذوزنقه‌های c، d، e و f را نشان می‌دهد.



(ت) (پ) (ب) (الف)

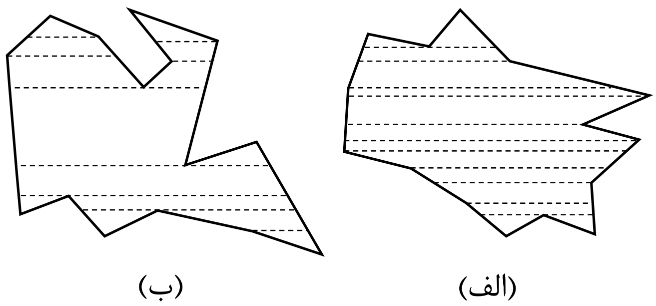
شکل ۳-۲) ناحیه آشکار از راس شماره ۱روی قاعده پایینی ذوزنقه‌های c، d، e و f

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 فرآیند خاتمه می‌یابد. اگر راس تفکیک در این بازه قرار نداشته باشد، مانند ذوزنقه f، 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[2] 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 برای راس تفکیک متوقف می‌شود.

[1] Visible
[2] Incremental

پیمایش چندضلعی‌ها به سمت پایین به منظور یافتن یک راس تفکیک، در بدترین حالت از مرتبه $O(n)$ است زیرا تعداد ذوزنقه‌های تولید شده از فرآیند تجزیه ذوزنقه‌ای از مرتبه $O(n)$ است [۱]. نتایج پیاده‌سازی نشان می‌دهد که در عمل تعداد ذوزنقه‌هایی که پیمایش می‌شوند از n بسیار کوچکتر است زیرا یا بعد از پیمایش چند ذوزنقه، طول بازه قابل دید از راس ادغام به صفر می‌رسد و یا پیمایش ذوزنقه‌ها به پایان می‌رسد.

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 عملیات جستجو برای یک راس تفکیک از مرتبه $O(1)$ است. شکل ظاهری چندضلعی در تعیین مقدار K اهمیت دارد. برای نشان دادن این موضوع در شکل ۳-۳ دو چندضلعی با تعداد راس‌های برابر که به صورت ذوزنقه‌ای تجزیه شده‌اند رسم شده است. در چندضلعی (الف) مقدار K باید حداقل برابر ۱۰ باشد اما برای چندضلعی (ب) مقدار K=۳ کفایت می‌کند.



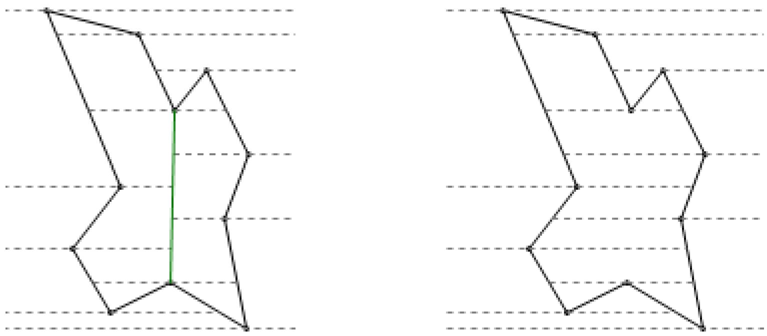
شکل ۳-۳) دو چندضلعی ساده با ۱۵ راس

در الگوریتم ۳-۲، 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.

```
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)))
end.
```

الگوریتم ۳-۲) اتصال یک راس ادغام به یک راس تفکیک

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 ۳-۴ یک چندضلعی را قبل و بعد از اجرای رویه HandleMergeVertex نشان می‌دهد.



(ب) (الف)

شکل ۳-۴) قبل و بعد از اضافه شدن یک قطر به چندضلعی

برای اتصال یک راس تفکیک به یک راس ادغام می‌توان مشابه با الگوریتم ۳-۲ 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.

در الگوریتم بهبود یافته، 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.

۳-۲-۲ الگوریتم بهبود یافته

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 [۹،۸،۷]. چنانچه از الگوریتم‌هایی که چندضلعی‌های حفره‌دار را به صورت ذوزنقه‌ای تجزیه می‌کنند استفاده شود، قادر به تجزیه یکنواخت یک چندضلعی حفره‌دار خواهیم بود. الگوریتمی که در ادامه ارائه می‌کنیم نسبت به الگوریتم ۳-۱ دارای دو مزیت است:

۱. کنترل زمان اجرا

با تعیین حداکثر عمق جستجو (K) می‌توان زمان اجرای الگوریتم را کنترل کرد. هر چه مقدار K به n میل کند، نتایج بهتری حاصل می‌شود (تعداد چندضلعی‌های کمتری تولید می‌شود) اما زمان اجرای رویه HandleMergeVertex نیز به $O(n)$ میل خواهد کرد. هرچه مقدار K کمتر شود، این رویه سریعتر اجرا می‌شود اما احتمال یافتن یک قطر ویژه نیز کمتر می‌شود. مقدار K با توجه به شرایط و کاربردهای مسئله انتخاب می‌شود. نتایج پیاده‌سازی الگوریتم پیشنهادی نشان می‌دهد که معمولا در عمل نیازی به استفاده از مقادیر بزرگ برای حداکثر عمق جستجو نیست و معمولا مقادیر ۲۰ و یا ۳۰ کفایت می‌کند.

۲. قابلیت تجزیه چندضلعی‌های حفره‌دار

الگوریتم ۳-۱ تنها قادر به تجزیه چندضلعی‌های ساده است. الگوریتمی که در اینجا معرفی می‌شود توانایی تجزیه چندضلعی‌های حفره‌دار را نیز دارد. برای انجام این کار، کافی است از الگوریتمی برای تجزیه ذوزنقه‌ای استفاده شود که قابلیت تجزیه چندضلعی‌های حفره‌دار را نیز داشته باشد.

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

الگوریتم ۳-۳) تجزیه یکنواخت یک چندضلعی حفره دار

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 ۳-۳ از رابطه زیر بدست می‌آید:

$$t(n, N_m, N_s) \in O(T) + O(n) + O(K\times N_m) + O(N_s) \tag{۵-۳}$$

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

$$t(n, N_m) \in O(T) + O(K\times N_m) \tag{۶-۳}$$

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

$$t(n, N_m) \in O(T) + O(N_m) \xRightarrow{N_m \in O(n) \text{ and } O(T) \in \Omega(n)} t(n, N_m) \in O(T) \tag{۷-۳}$$

بنابراین اگر K یک مقدار ثابت باشد، هزینه کلی الگوریتم برابر با هزینه تجزیه ذوزنقه‌ای چندضلعی P است. در بخش بعد نشان می‌دهیم که در عمل با مقادیر ثابت و نسبتا کوچک K می‌توان به نتایج مطلوبی دست یافت. اگر مقدار K متناسب با مقدار n انتخاب شود، یعنی $K \in O(n)$ باشد، آنگاه هزینه کلی الگوریتم ۳-۳ از مرتبه زیر است:

$$t(n, N_m) \in O(T) + O(n \times N_m) \quad (۹-۳)$$

۳-۳ پیاده‌سازی و مقایسه عملی 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[۱] الگوریتم تصادفی افزایشی *سایدل* استفاده کرده‌ایم [۷]. زمان اجرای مورد انتظار 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 انجام آزمایش‌ها حداکثر عمق جستجو را مقداری ثابت در نظر گرفته‌ایم، بنابراین 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 جدول ۳-۱ نتایج حاصل از مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی* درج شده است. برای مقایسه این الگوریتم‌ها با یکدیگر از یک پایگاه داده حاوی ۱۶۰۰ چندضلعی ساده که

[۱] سایدل ابتدا یک الگوریتم اولیه با زمان اجرای مورد انتظار $O(n \log n)$ برای تجزیه ذوزنقه‌ای ارائه کرد. سپس با بهبود الگوریتم اولیه زمان اجرای آن را به $O(n \log^* n)$ کاهش داد [۲۲].

به صورت تصادفی تولید شده‌اند استفاده شده است[۱]. تعداد راس‌های چندضلعی‌های تولید شده در ستون اول جدول ۳-۱ ملاحظه می‌شود. به ازای هر مقدار مانند v که در این ستون درج شده است، یک مجموعه با ۱۰۰ چندضلعی ساده که هر کدام از آنها v راس دارد تولید شده است. ستون‌های دوم تا ششم، 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).
با توجه به اینکه عملکرد الگوریتم حریصانه با افزایش حداکثر عمق جستجو (K) بهبود می‌یابد، چهار نسخه از این الگوریتم با مقادیر مختلف K را مورد مقایسه و ارزیابی قرار داده‌ایم.

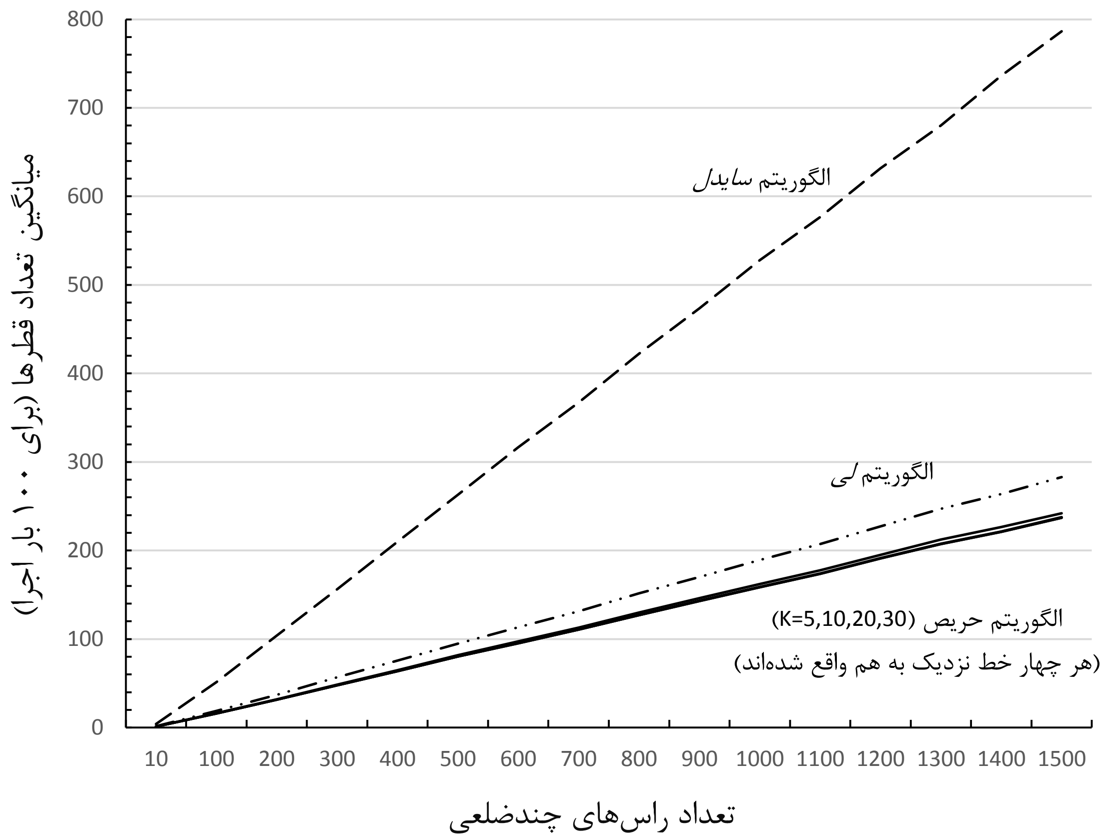
جدول ۳-۱) مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی*

تعداد راس های چندضلعی (n)	**میانگین تعداد قطر الگوریتم *سایدل***	**میانگین تعداد قطر الگوریتم *لی***	**میانگین تعداد قطر الگوریتم پیشنهادی**			
			K=5	K=10	K=20	K=30
10	4.13	1.36	1.23	1.23	1.23	1.23
100	51.4	18.73	16.25	15.93	15.90	15.90
200	103.91	37.02	31.94	31.48	31.43	31.43
300	156.44	56.69	48.68	47.73	47.66	47.65
400	209.72	75.57	64.90	63.87	63.72	63.72
500	262.88	94.83	81.79	80.44	80.35	80.35
600	316.60	113.45	97.38	95.54	95.34	95.33
700	367.11	131.44	112.81	110.89	110.68	110.68
800	422.09	151.52	129.74	127.06	126.88	126.88
900	473.63	170.18	146.17	143.34	143.04	143.04
1000	527.80	189.07	162.14	158.74	158.43	158.43
1100	576.71	207.19	177.55	173.98	173.58	173.53
1200	631.74	227.14	195.07	191.37	190.90	190.89
1300	680.12	247.21	212.49	207.83	207.34	207.34
1400	736.12	263.67	226.66	221.48	221.07	221.06
1500	786.32	282.87	242.20	237.56	237.15	237.15

[۱] چندضلعی‌های تصادفی را با استفاده از کتابخانه CGAL تولید کردیم.

برای مقایسه آسان‌تر این سه الگوریتم، داده‌های جدول ۳-۱ در نمودار ۳-۱ ترسیم شده است. همانطور که از این نمودار ملاحظه می‌شود، الگوریتم پیشنهادی به نسبت الگوریتم *سایدل* و الگوریتم *لی* از تعداد قطرهای کمتری برای تجزیه یکنواخت استفاده می‌کند. برای ارزیابی دقیق‌تر عملکرد الگوریتم پیشنهادی، از چهار مقدار ۵، ۱۰، ۲۰ و ۳۰ به عنوان حداکثر عمق جستجو استفاده شده است. از این نمودار می‌توان دریافت که با انتخاب مقادیر کوچک (و ثابت) به عنوان حداکثر عمق جستجو می‌توان به نتایج مطلوب دست یافت (حداقل در مقایسه با الگوریتم *سایدل* و الگوریتم *لی*).

نمودار ۳-۱) مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی*



فصل ۴

نتیجه‌گیری و پیشنهادات

در این فصل به مرور کلی تحقیق و نتیجه‌گیری در مورد نحوه عملکرد الگوریتم پیشنهادی ارائه شده در فصل قبل پرداخته می‌شود. در بخش ۴-۲ پیشنهاداتی برای بهبود عملکرد این الگوریتم ارائه شده است.

۴-۱ مرور تحقیق

تجزیه چندضلعی یکی از مباحث مهم و با سابقه طولانی در حوزه هندسه محاسباتی است. در اکثر موارد پردازش چندضلعی‌هایی که یکنواخت یا محدب نیستند، دشوارتر از چندضلعی‌هایی است که این ویژگی‌ها را دارند. به همین دلیل در اغلب کاربردهای عملی، ابتدا چندضلعی به صورت مثلثی، ذوزنقه‌ای، محدب، یکنواخت، ستاره‌ای و یا هر شکل هندسی دیگری که نیازهای مسئله را برآورده کند تجزیه می‌شود. تصمیم‌گیری در مورد نوع تجزیه (محدب، یکنواخت و غیره) وابسته به شرایط مسئله است.

در این پایان‌نامه به مطالعه و بررسی تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار پرداخته شده است. در فصل اول به برخی از مفاهیم مقدماتی از حوزه هندسه محاسباتی که در فصل‌های دوم و سوم به آنها نیاز است، اشاره شده است. در فصل دوم به مطالعه الگوریتم‌های *لی*، *لیو* و *سایدل* پرداخته شده است. الگوریتم‌های لی و لیو تنها قادر به تجزیه چندضلعی‌های ساده می‌باشند اما می‌توان الگوریتم سایدل را برای تجزیه چندضلعی‌های حفره‌دار نیز بکار برد. در فصل سوم یک الگوریتم حریصانه جهت تجزیه یکنواخت چندضلعی‌های ساده و حفره‌دار ارائه شده است. این الگوریتم برای تجزیه چندضلعی از نقاط کمکی استفاده نمی‌کند. در بخش ۳-۳ نتایج حاصل از پیاده‌سازی الگوریتم پیشنهادی و مقایسه آن با الگوریتم لی و سایدل بحث و بررسی شده است.

۲-۴ نتیجه‌گیری

تا کنون پنج الگوریتم برای تجزیه یکنواخت چندضلعی‌های ساده و یا حفره‌دار وجود دارد [۲۷]. جزئیات مربوط به این الگوریتم‌ها در جدول ۳-۱ درج شده است. الگوریتم‌های *لیو* و *نتافوس*، *وی* و *کیل* چندضلعی را به صورت کمینه تجزیه می‌کنند اما الگوریتم‌های لی و سایدل تجزیه را الزاما به صورت کمینه انجام نمی‌دهند. زمان اجرای الگوریتم کیل از مرتبه $O(Nn^4)$ و الگوریتم لیو 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. یادآوری می‌شود که تجزیه کمینه یک چندضلعی حفره‌دار بدون استفاده از نقاط کمکی، یک مسئله NP-سخت است [۴].

هدف از الگوریتمی که در این پایان‌نامه ارائه شده است، بوجود آوردن تعادلی بین زمان اجرا و تجزیه کمینه است. در فصل سوم ثابت شد که زمان اجرای این الگوریتم در حالتی که حداکثر عمق جستجو یک مقدار ثابت است، از مرتبه $O(T)$ می‌باشد (T زمان مورد نیاز برای تجزیه ذوزنقه‌ای چندضلعی است). با توجه به اینکه برای تجزیه ذوزنقه‌ای یک چندضلعی، الگوریتم‌های کارآیی موجود است، می‌توان به سرعت یک چندضلعی ساده و یا حفره‌دار را بدون استفاده از نقاط کمکی توسط الگوریتم پیشنهادی به صورت یکنواخت تجزیه کرد. 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).

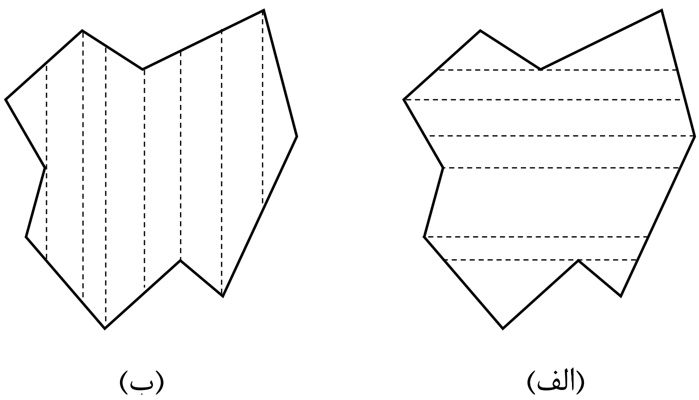
این الگوریتم نسبت به الگوریتم لی نیز از تعداد قطرهای کمتری برای تجزیه چندضلعی استفاده می‌کند (به طور میانگین کاهش ۱۷ درصدی در استفاده از قطرها برای تجزیه چندضلعی‌هایی با ۱۵۰۰ راس).

۴-۲ کارهای آینده

برای ادامه این تحقیق تصمیم بر این است که بهبودهایی بر الگوریتم پیشنهادی اعمال شود و عملکرد آن به طور دقیق‌تر مورد مطالعه قرار گیرد. در این راستا دو فعالیت اصلی مد نظر قرار دارد که در ادامه به تشریح آنها پرداخته می‌شود.

۴-۲-۱ انتخاب بین تجزیه ذوزنقه‌ای افقی یا عمودی

در گام اول الگوریتم ۳-۳ چندضلعی ورودی به صورت افقی تجزیه ذوزنقه‌ای می‌شود. با اعمال تغییراتی در این الگوریتم می‌توان از تجزیه ذوزنقه‌ای عمودی (به جای تجزیه افقی) استفاده کرد. در بعضی از چندضلعی‌ها، تجزیه ذوزنقه‌ای عمودی منجر به تولید جواب بهتری می‌شود (با استفاده از تعداد قطرهای کمتری می‌توان چندضلعی را تجزیه کرد و یا اینکه نیاز به پیمایش چندضلعی‌های کمتری است). شکل ۴-۱ چنین وضعیتی را نشان می‌دهد.



شکل ۴-۱) تجزیه ذوزنقه‌ای افقی و عمودی یک چندضلعی ساده

برای دست یافتن به یک قطر ویژه در چندضلعی شکل ۴-۱(الف) که به صورت افقی تجزیه ذوزنقه‌ای شده است، نیاز به پیمایش پنج ذوزنقه وجود دارد در حالی که در تجزیه عمودی تنها نیاز به پیمایش یک ذوزنقه وجود دارد. بنابراین در این شرایط بهتر است چندضلعی به صورت عمودی تجزیه شود و

نه افقی. اگر بتوان قبل از تجزیه ذوزنقه‌ای چندضلعی را به سرعت پیش‌پردازش[1] کرد (بهتر است این عمل در زمان در $O(n)$ انجام شود) و در مورد افقی یا عمودی بودن تجزیه تصمیم گرفت، می‌توان به نتایج بهتری دست یافت.

۴-۲-۲ پیش‌پردازش قطرها

یکی از تفاوت‌های الگوریتم پیشنهادی با الگوریتم *لیو* این است که الگوریتم *لیو* از میان مجموعه تمامی قطرهای ویژه‌ای که می‌توان به چندضلعی اضافه کرد، زیر مجموعه‌ای را محاسبه می‌کند که موجب تجزیه کمینه می‌شود اما الگوریتم پیشنهادی (به دو دلیل که در بخش ۳-۱ بحث شد) این زیرمجموعه را محاسبه نمی‌کند و به صورت حریصانه هر زمان که امکان اضافه کردن یک قطر ویژه وجود داشت، آن را به چندضلعی اضافه می‌کند. باید توجه داشت که در بعضی از شرایط نباید یک قطر ویژه را اضافه کرد زیرا آن قطر مانع اضافه شدن قطرهای دیگر به چندضلعی می‌شود. محاسبه دقیق زیرمجموعه‌ای از قطرها که موجب تجزیه کمینه می‌شود عملی زمان‌بر است. اگر این امکان وجود داشته باشد که با استفاده از بعضی روش‌های محاسباتی، زیرمجموعه‌ای از قطرهای ویژه را به سرعت محاسبه کرد که منجر به تولید چندضلعی‌های یکنواخت کمتری شود، به جواب بهینه نزدیکتر شده‌ایم.

[1] preprocess

مراجع

[1] Berg, M.D., Cheong, O., Kreveld, M.V. and Overmars, M. 2008. Computational Geometry: Algorithms and Applications, Springer.

[2] Preparata, F.P. and Shamos, M.I. 1985. Computational Geometry: An Introduction. Springer-Verlag.

[3] Chen, J. 1996. Computational Geometry: Methods and Applications. Texas A&M University.

[4] Keil, J.M. 1996. Polygon Decomposition. Department of Computer Science, University of Saskatchewan, Saskatoon Sask, Canada.

[5] Liu, R. and Ntafos, S. 1988. On decomposing polygons into uniformly monotone parts. Information Processing Letters, 27: 85-89.

[6] O'Rourke, J. 1990. Computational Geometry in C. Cambridge University Press.

[7] 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.

[8] Lorenzetto, G.P. and Datta, A. 2002. A linear time heuristics for trapezoidation of GIS polygons. ICCS, LNCS 2331, 75−84.

[9] Zalik, B., Clapworthy, G.J. 1999. A universal trapezoidation algorithm for planar polygons. Computers & Graphics, 23: 353-363.

[10] El Gindy, H. and Avis, D. 1981. A linear algorithm for computing the visibility polygon from a point. Journal of algorithms, 2: 186-197.

[11] Lee, D.T. 1983. Visibility of a simple polygon. Computer Vision, Graphics and Image Processing, 22: 207–221.

[12] Asano, T. 1985. An Efficient Algorithm for Finding the Visibility Polygon for a Polygonal Region with Holes. IEICE Transactions, 68(9): 557-559.

[13] Suri, S. and O'Rourke, J. 1986. Worst-Case optimal algorithms for constructing visibility polygons with holes. In Proc. of the second annual symposium on Computational Geometry, 14-23.

[14] Dehne, F., Sack, J.R. and Santoro, N. 1995. An Optimal Algorithm for Computing Visibility in the Plane. SIAM Journal on Computing, 24(1): 184–201.

[15] Zarei, A.R. and Ghodsi, M. 2005. Efficient computation of query point visibility in polygons with holes. In Proc. 21st Annual Symposium on Computational Geometry, 314-320.

[16] Sack, J.R. and Urrutia, J. 2000. Handbook of computational geometry. Elsevier Science.

[17] Supowit, K.J. 1987. Finding a maximum planar subset of a set of nets in a channel. IEEE Transactions On Computer-Aided Design, 6(1).

[18] Nash, N. and Gregg, D. 2010. An output sensitive algorithm for computing a maximum independent set of a circle graph. Information Processing Letters, 110: 630–634.

[19] Apostolico, A., Atallah, M.J. and Hambrusch, S.E. 1992. New clique and independent set algorithms for circle graphs. Discrete Applied Mathemathic, 36(1): 1–24.

[20] Valiente, G. 2003. A new simple algorithm for the maximum-weight independent set problem on circle graphs. Lecture Notes in Computer Science, 2906: 129-137.

[21] Nash, N., Lelait, S. and Gregg D. 2009. Efficiently implementing maximum independent set algorithms on circle graphs. Journal of Experimental Algorithmics, 13: 1-34.

[22] Asano, T., Imai, H. and Mukaiyama, A. 1991. Finding a maximum weight independent set of a circle graph. IEICE Transactions, 74(4): 681–683.

[23] Gavril, F. 1973. Algorithms for a maximum clique and a maximum independent set of a circle graph. Networks, 3(3): 261–273.

[24] Spinrad, J. 1994. Recognition of Circle graphs. Journal of algorithms, 16: 264-282.

[25] Lee, D.T. and Preparata, F.P. 1977. Location of a point in a planar subdivision and its applications. SIAM Journal on Computing, 6: 594–606.

[26] Chazeel, B. 1991. Triangualting a Polygon in Linear Time. Discrete Computational Geometry, 6: 485-524.

[27] Wei, X., Joneja, A. and Mount, D.M. 2012. Optimal uniformly monotone partitioning of polygons with holes. Computer-Aided Design, 44: 1235–1252.

[28] Subramaniam, L. 2003. Partition of a non-simple polygon into simple polygons. Master of Science Thesis. University of South Alabama.

Surname: Tajedini			Firstname: Ahmad

Thesis Title: An Algorithm for Monotone Decomposition of Polygons in 2D Space

Supervisor: Dr. J. Karimpour
Advisor: Dr. Sh. Lotfi

Degree: MSc Major: Computer Science Field: Computer Systems University: Tabriz

Faculty: Mathematical Science	Graduation Date: September 11, 2013	Pages: 65

Keywords: polygon decomposition, monotone polygon, simple polygon, polygon with hole, greedy algorithm, computational geometry

Abstract:

Partitioning a polygon into a set of smaller polygons is called polygon decomposition. There are several methods to decompose a polygon: convex decomposition, monotone decomposition and trapezoidal decomposition, which all of them commonly used in computational geometry. In some cases, it is necessary or desirable to generate minimum number of sub polygons.

In this desertation, we propose a greedy algorithm to monotone decomposition of polygons with holes. This algorithm doesn't use Steiner points. The main goal of developing this algorithm is achieving a near minimum decomposition at an acceptable time. Since minimum decomposition of a polygon with holes when Steiner points are not allowed, is a NP-Hard problem, near optimum solutions are the only available practical options for large instances of the problem. In developing this algorithm, two issues have been considered. First subject is about minimality of decomposition. However there is no guarantee about getting the minimum deal, the results of practical implementations demonstrate the effectiveness of this approach. The second issue which is considered in designing of this algorithm is the run time matter. A part of this algorithm is controlled by using a parameter which called ‘Maximum Search Depth’. As the value of this parameter is smaller, the probability of finding near minimum results will be decreased, and in the same, the run time of the algorithm will be decreased. By assigning larger values to this parameter, we can produce better answers but the algorithm's run time will be increased too. By setting this parameter according to the application of this algorithm, we can switch and select between the minimality of analysis or run time performance, or create a balance between them.

University of Tabriz
Faculty of Mathematics
Department of Computer Science

Submitted in Partial Fulfillment of the Requirements for M. Sc. Degree in Computer Science

Title

An Algorithm for Monotone Decomposition of Polygons in 2D Space

Supervisor

Dr. J. Karimpour

Advisor

Dr. Sh. Lotfi

Researcher

A. Tajedini

Date: September 2013