

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 1392

**بسم اللّه الرحمن الرحیم**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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: 20/6/1392 | | | تعداد صفحات: 65 | | |
| 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 انسانی­شان از کلمه ایثار و از خودگذشتگی

به پاس عاطفه سرشار و گرمای امید بخش وجودشان

به پاس قلب­های بزرگشان که فریادرس است و سرگردانی و ترس در پناهشان به شجاعت می­گراید

و به پاس محبت­های بی­ دریغشان که هرگز فروکش نمی­کند

این مجموعه را تقدیم می­کنم 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 کریم­پور و جناب آقای دکتر لطفی

چرا که بدون راهنمایی­های ارزنده ایشان تهیه این پایان­نامه بسیار مشکل می­نمود

**فهرست مطالب**

|  |  |
| --- | --- |
| عنوان | شماره صفحه |

[**فصل 1** مقدمه 1](#_Toc365360792)

[1-1 اصطلاحات 3](#_Toc365360793)

[1-2 بیان مسئله 4](#_Toc365360794)

[1-3 نظریه 4](#_Toc365360795)

[1-4 اهداف پایان­نامه 5](#_Toc365360796)

[1-5 سازمان پایان­نامه 5](#_Toc365360797)

[**فصل 2** پیشینه­های پژوهشی 7](#_Toc365360798)

[2-1 تعاریف اولیه و مفاهیم مقدماتی 10](#_Toc365360799)

[2-1-1 انواع چندضلعی 11](#_Toc365360800)

[2-1-2 انواع راس­ها در چندضلعی 14](#_Toc365360801)

[2-1-3 نمایش چندضلعی و زیر فضا در حافظه 17](#_Toc365360802)

[2-1-4 تجزیه ذوزنقه­ای یک چندضلعی ساده 19](#_Toc365360803)

[2-1-5 قضیه چندضلعی یکنواخت قائم 21](#_Toc365360804)

[2-2 الگوریتم لی و پریپاراتا 26](#_Toc365360805)

[2-3 الگوریتم لیو و نتافوس 33](#_Toc365360806)

[2-4 الگوریتم مثلثی‏سازی سایدل 37](#_Toc365360807)

[2-4-1 استفاده از الگوریتم سایدل برای تجزیه یکنواخت یک چندضلعی 38](#_Toc365360808)

[2-4 جمع­بندی و نتیجه­گیری 39](#_Toc365360809)

[**فصل 3** ارائه یک الگوریتم حریصانه برای تجزیه یکنواخت چندضلعی 43](#_Toc365360810)

[3-1 الگوریتم اولیه 44](#_Toc365360811)

[3-1-1 چندضلعی آشکار از یک نقطه 21](#_Toc365360812)

[3-1-2 الگوریتم اولیه 45](#_Toc365360813)

[3-2 بهبود الگوریتم اولیه 48](#_Toc365360814)

[3-2-1 حذف راس­های ادغام و تفکیک به کمک تجزیه ذوزنقه­ای 48](#_Toc365360815)

[3-2-2 الگوریتم بهبود یافته 54](#_Toc365360816)

[3-3 پیاده­سازی و مقایسه عملی با الگوریتم­های موجود 56](#_Toc365360817)

[**فصل 4** نتیجه­گیری و پیشنهادات 59](#_Toc365360818)

[4-1 مرور تحقیق 60](#_Toc365360819)

[4-2 نتیجه­گیری 61](#_Toc365360820)

[4-2 کارهای آینده 62](#_Toc365360821)

[4-2-1 انتخاب بین تجزیه ذوزنقه­ای افقی یا عمودی 62](#_Toc365360822)

[4-2-2 پیش­پردازش قطرها 63](#_Toc365360823)

**فهرست جدول­ها**

|  |  |
| --- | --- |
| عنوان | شماره صفحه |

[جدول 1-1) اصطلاحات بکار رفته در این پایان­نامه 3](#_Toc365972135)

[جدول 2-1) برخی از الگوریتم­های شناخته شده برای تجزیه یکنواخت یک چندضلعی 41](#_Toc365972136)

[جدول 3-1) مقایسه الگوریتم پیشنهادی با الگوریتم سایدل و الگوریتم لی 57](#_Toc365972137)

**فهرست شکل­ها**

|  |  |
| --- | --- |
| عنوان | شماره صفحه |

[شکل 2-1) چند مثال از چندضلعی ساده و غیر ساده 11](#_Toc365635301)

[شکل 2-2) برخی از ویژگی­های چندضلعی محدب 12](#_Toc365635302)

[شکل 2-3) یک چندضلعی ساده که نسبت به محور Y یکنواخت است 13](#_Toc365635303)

[شکل 2-4) یک چندضلعی یکنواخت قائم 14](#_Toc365635304)

[شکل 2-5) انواع راس­های یک چندضلعی ساده 15](#_Toc365635305)

[شکل 2-6) یک چندضلعی ساده که به چندضلعی­های یکنواخت قائم تجزیه شده است. 17](#_Toc365635306)

[شکل 2-7) نمایش چندضلعی شکل 2-6 در حافظه 18](#_Toc365635307)

[شکل 2-8) تجزیه ذوزنقه­ای یک چندضلعی ساده 19](#_Toc365635308)

[شکل 2-9) چندضلعی آشکار از نقطه a در یک چندضلعی ساده 22](#_Toc365635309)

[شکل 2-10) یک چندضلعی ساده که یکنواخت قائم نیست. 23](#_Toc365635310)

[شکل 2-11) دو حالت امکان­پذیر در قضیه چندضلعی یکنواخت قائم 24](#_Toc365635311)

[شکل 2-12) یک مثال برای نشان دادن اینکه قضیه 1-1 در جهت عکس برقرار نیست. 25](#_Toc365635312)

[شکل 2-13) مراحل تجزیه یک چندضلعی ساده توسط الگوریتم *لی* و *پریپاراتا* 26](#_Toc365635313)

[شکل 2-14) پردازش یک راس تفکیک در الگوریتم *لی و پریپاراتا* 29](#_Toc365635314)

[شکل 2-15) پردازش یک راس ادغام 29](#_Toc365635315)

[شکل 2-16) یک گراف و چهار مجموعه مستقل آن 33](#_Toc365635316)

[شکل 2-17) مراحل تجزیه یکنواخت یک چندضلعی ساده با استفاده از الگوریتم سایدل 38](#_Toc365635317)

[شکل 3-1) یک چندضلعی ساده که به صورت ذوزنقه­ای تجزیه شده است. 48](#_Toc365635318)

[شکل 3-3) دو چندضلعی ساده با 15 راس 51](#_Toc365635319)

[شکل 3-4) قبل و بعد از اضافه شدن یک قطر به چندضلعی 53](#_Toc365635320)

[شکل 4-1) تجزیه ذوزنقه‌ای افقی و عمودی یک چندضلعی ساده 62](#_Toc365635321)

# 

# فصل 1 مقدمه

تجزیه چندضلعی[[1]](#footnote-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]](#footnote-2) و­ یا نسبت به یک خط خاص یکنواخت[[3]](#footnote-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]](#footnote-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]](#footnote-5) برای تجزیه چندضلعی استفاده نمی­کند.

## 1-1 اصطلاحات

اصطلاحات بکار رفته در این پایان­نامه در جدول 1-1 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.

|  |  |
| --- | --- |
| 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 180 درجه است و پایین­تر از راس­های مجاور خود واقع شده است. |
| راس تفکیک | راسی که زاویه داخلی آن بیشتر از 180 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. |

جدول 1-1) اصطلاحات بکار رفته در این پایان­نامه

## 1-2 بیان مسئله

در این تحقیق به مطالعه و بررسی الگوریتم­های موجود و همچنین ارائه یک الگوریتم جهت تجزیه یکنواخت[[6]](#footnote-6) 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[[7]](#footnote-7). 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.

## 1-3 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[[8]](#footnote-8). 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.

## 1-4 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.

## 1-5 سازمان پایان­نامه

پس از مقدمه، در فصل 2 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 3 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 2 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[[9]](#footnote-9)، یکنواخت[[10]](#footnote-10)، محدب[[11]](#footnote-11)، ستاره­ای[[12]](#footnote-12) و  
ذوزنقه­ای[[13]](#footnote-13) ارائه شده است [4،3،2،1]. 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 [4]. دلیل این موضوع این است که بسیاری از الگوریتم­ها روی این نوع چندضلعی­ها به صورت کارآتری اجرا می­شوند [6،5،3،1].

در برخی از کاربردها، لازم و یا مطلوب است که تجزیه به صورت کمینه انجام شود [5،4].  
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[[14]](#footnote-14) 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 که مقدار زاویه داخلی آنها بیشتر از 180 درجه است. به این راس­ها، راس­ بازتابی[[15]](#footnote-15) یا راس شکافی[[16]](#footnote-16) گفته می­شود. عامل سوم تعداد حفره­های[[17]](#footnote-17) 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[[18]](#footnote-18) 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[[19]](#footnote-19)* 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*[[20]](#footnote-20) یک الگوریتم با زمان اجرای O(Nn4) و لیو و نتافوس یک الگوریتم با زمان اجرای O(nN3 + N2n log n + N5)جهت تجزیه کمینه یک چندضلعی ساده ارائه داده­اند [5،4]. چنانچه استفاده از نقاط کمکی مجاز باشد، الگوریتم لیو و نتافوس (با اعمال تغییراتی) به یک الگوریتم با پیچیدگی محاسباتی  
O(n N3 log n + N5) تبدیل می­شود. *کیل* همچنین یک الگوریتم از مرتبه O(Nn4) 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[[21]](#footnote-21)* یک الگوریتم با زمان اجرای O(K(n log n + hlog3 h)) برای تجزیه کمینه یک چندضلعی حفره­دار با استفاده از نقاط کمکی، ارائه کرده است [27]. در عبارت اخیر، K تعداد اضلاع گراف آشکاری[[22]](#footnote-22) 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-سختاست [4].

در این فصل، بعد از مرور برخی تعاریف و مفاهیم مقدماتی مورد نیاز، الگوریتم­های تجزیه لی و پریپاراتا[[23]](#footnote-23)، لیو و نتافوس و سایدل مورد مطالعه و بررسی قرار می­گیرد. این الگوریتم­ها یک چندضلعی را به مجموعه­ای از چندضلعی­های یکنواخت قائم تجزیه می­کنند.

## 2-1 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.

### 2-1-1 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[[24]](#footnote-24). 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 [4،1]. 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 2-1 مثال­هایی از چندضلعی­های ساده و غیر ساده را نشان می­دهد.

**(الف)**

**(ب)**

**(پ)**

شکل 2-1) 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 180 درجه است، چندضلعی محدب (کوژ) نامیده می­شود. اگر دو نقطه دلخواه درون یک چندضلعی محدب با یک پاره­خط به یکدیگر متصل 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 180 درجه باشند. چندضلعی­های محدب دارای ویژگی­های جالبی می­باشند و در هندسه محاسباتی مورد توجه قرار دارند. یکی از این ویژگی­ها، سهولت مثلثی‏سازی[[25]](#footnote-25) 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.

b

a

**(الف)**

**(ب)**

شکل 2-2) 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[[26]](#footnote-26) 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[[27]](#footnote-27).

A1

A2

A3

A4

**A**

**B**

**P**

**C**

C1

شکل 2-3) یک چندضلعی ساده که نسبت به محور Y یکنواخت است

در شکل 2-3 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 A4) و یا فقط یک بار سطح چندضلعی را قطع می­کند (مانند خط­های A1، A2 و A3). 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 C1 (که بر C عمود است) و چندضلعی P یک پاره خط پیوسته نیست بلکه از اشتراک این دو، سه پاره­خط به وجود آمده است (C1 سه بار سطح P را قطع کرده است).

یک چندضلعی که نسبت به یک خط موازی با محور Y یکنواخت باشد، چندضلعی یکنواخت قائم[[28]](#footnote-28) 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 2-4 نشان داده شده است.

پایین‎ترین راس

Y

X

بالاترین راس

جهت پیمایش در زنجیره‎های راست و چپ، همواره به سمت پایین و یا افقی است.

زنجیره راس‌های سمت راست

زنجیره راس‌های سمت چپ

**P**

شکل 2-4) 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[[29]](#footnote-29). یکنواخت بودن نسبت به تعداد بی­شماری خط باعث محدود شدن شکل چندضلعی­های یکنواخت به اشکال تقریبا حلقوی (دایره­ای) شکل می­شود.

### 2-1-2 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 2-5 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.

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

P

1

راس معمولی

راس شروع

راس پایان

راس تفکیک

راس ادغام

شکل 2-5) انواع راس­های یک چندضلعی ساده

در چندضلعی P، راس شماره 1 بالاترین راس است. همچنین با وجود اینکه مؤلفه Y راس­های شماره 6 و 7 با هم برابر است، اما راس شماره 7 بالاتر از راس شماره 6 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 9 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 2 و 19 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[[30]](#footnote-30) نامیده می­شود. در چندضلعی P، راس­های شماره 1، 2، 3، 6، 8، 9، 13، 14، 15، 16، 18 و 19 راس­­های چرخشی هستند.

راسی که چرخشی نباشد، راس معمولی[[31]](#footnote-31) 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:

1. راس شروع[[32]](#footnote-32)  
   راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن کمتر از 180 درجه است، راس شروع نامیده می­شود. در چندضلعی P، راس­های شماره 1، 3، 13 و 18 راس­های شروع می­باشند.
2. راس پایان[[33]](#footnote-33)  
   راسی که از دو راس مجاور خود پایین­تر و زاویه داخلی آن کمتر از 180 درجه است، راس پایان نامیده می­شود. در چندضلعی P، راس­های شماره 6، 9 و 16 راس­های پایان می­باشند.
3. راس ادغام[[34]](#footnote-34)  
   راسی که از دو راس مجاور خود پایین­تر و زاویه داخلی آن بیشتر از 180 درجه است، راس ادغام نامیده می­شود. در چندضلعی P، راس­های شماره 2، 4 و 19 راس­های ادغام می­باشند.
4. راس تفکیک[[35]](#footnote-35)  
   راسی که از دو راس مجاور خود بالاتر و زاویه داخلی آن بیشتر از 180 درجه است، راس تفکیک نامیده می­شود. در چندضلعی P، راس­های شماره 8 و 15 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 2-5) 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 8 به دو قسمت تقسیم (تفکیک) می­شود.  
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 1 و 3 تولید می­شود و به سمت پایین چندضلعی P در حرکت است. این دو جریان مختلف آب، در راس شماره 2 با هم ادغام می­شوند.

### 2-1-3 نمایش چندضلعی و زیر فضا در حافظه

برای پیاده­سازی الگوریتم‏های هندسه محاسباتی، در ابتدا باید اشکال هندسی را در قالب ساختمان­ داده­های[[36]](#footnote-36) 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[[37]](#footnote-37) ذخیره کرد. هر چند این روش برای نمایش یک چندضلعی ساده مناسب است، اما قادر به نمایش یک چندضلعی تجزیه شده (زیر فضا[[38]](#footnote-38)) 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 2-6 یک چندضلعی تجزیه شده را نشان می­دهد.

1

2

3

4

5

6

7

8

9

10

11

12

P1

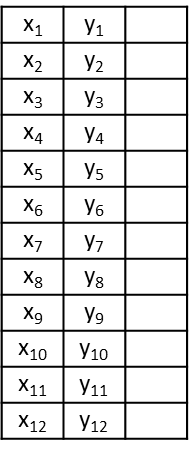
P2

P3

P4

شکل 2-6) یک چندضلعی ساده که به چندضلعی­های یکنواخت قائم تجزیه شده است.

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 2-7 نحوه ذخیره­سازی چندضلعی ترسیم شده در شکل 2-6 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.



2

5

5

9

11

11

ستون اشاره­گرها

مختصات راس­های چندضلعی

لیست راس­های همجوار قطری

شکل 2-7) نمایش چندضلعی شکل 2-6 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.

### 2-1-4 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[[39]](#footnote-39) نامیده می­شود[[40]](#footnote-40).

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 2-8 یک چندضلعی که به صورت افقی و عمودی تجزیه شده است را نشان می­دهد.

**1**

**8**

**7**

**m**

**j**

**i**

**(الف)**

**(ب)**

**n**

**p**

**2**

**3**

**4**

**5**

**6**

شکل 2-8) تجزیه ذوزنقه­ای یک چندضلعی ساده. (الف) تجزیه ذوزنقه‌ای افقی  
(ب) تجزیه ذوزنقه‌ای عمودی

چندضلعی شکل 2-8 (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 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 5) دقیقا دارای دو همسایه بالایی می­باشند (ذوزنقه­های شماره 3 و 4). به همین ترتیب ذوزنقه­هایی که یک راس تفکیک روی قاعده پایینی آنها قرار گرفته است (ذوزنقه شماره 6) دقیقا دارای دو همسایه پایینی می­باشند (ذوزنقه­های شماره 7 و 8).

تجزیه ذوزنقه­ای در سال 1984 توسط *چازل* و *اینسرپی*[[41]](#footnote-41) ابداع شد. آنها از این نوع تجزیه برای حل مسئله مثلثی‏سازی استفاده کردند [6]. تجزیه ذوزنقه­ای کاربردهای دیگری نیز دارد که از میان آنها می­توان به برنامه­ریزی حرکت ربات[[42]](#footnote-42) و مسئله موقعیت نقطه اشاره[[43]](#footnote-43) کرد [1]. تا کنون الگوریتم­های مختلفی برای تجزیه ذوزنقه­ای یک چندضلعی ارائه شده است. *سایدل*[[44]](#footnote-44) یک الگوریتم تصادفی افزایشی[[45]](#footnote-45) برای حل این مسئله ارائه کرده است. زمان اجرای مورد انتظار[[46]](#footnote-46) این الگوریتم از مرتبه O(n log\* n)  
و پیچیدگی مورد انتظار حافظه مصرفی[[47]](#footnote-47) آن از مرتبه O(n) است [7]. 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 ≥ 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  
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 [8]. *زالیک* و *کلاپورسی* نیز یک الگوریتم با زمان اجرای  
O(n2 log n) برای تجزیه ذوزنقه­ای یک چندضلعی حفره دار ارائه کرده­اند [9].

### 2-1-5 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[[48]](#footnote-48) اگر و تنها اگر پاره­خط به طور کامل درون P واقع شود. به عبارت دیگر:

نقاط a و b یکدیگر را می­بینند

به مجموع تمام نقاط داخل چندضلعی P که از نقطه a آشکار است، چندضلعی آشکار از نقطه a گفته می­شود[[49]](#footnote-49). چندضلعی آشکار از یک نقطه، یک چندضلعی ساده است. برای محاسبه چندضلعی آشکار کافی است مختصات راس­های آن را بدست آورد. در شکل 2-9 یک چندضلعی ساده و یک چندضلعی حفره­دار رسم شده است. چندضلعی آشکار از یک نقطه دلخواه درون این چندضلعی با نقطه­چین مشخص شده است.

**(الف)**

**(الف)**

**(ب)**

a

a

شکل 2-9) 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*[[50]](#footnote-50) مجموع تمام نقاطی از چندضلعی است که از آن ضلع آشکار است. *گیندی*[[51]](#footnote-51) در سال 1981 یک الگوریتم خطی (بهینه) برای محاسبه چندضلعی آشکار از یک نقطه در یک چندضلعی ساده ارائه کرد [10].

دو سال بعد، *لی*[[52]](#footnote-52) نیز یک الگوریتم خطی دیگر برای محاسبه چندضلعی آشکار از یک نقطه (داخل و بیرون چندضلعی) ارائه کرد [11]. الگوریتم­های خطی دیگری نیز برای این مسئله ارائه شده است [27،26]. از میان الگوریتم­های مختلفی که برای محاسبه چندضلعی آشکار در چندضلعی­های حفره‌دار ارائه شده است می­توان به الگوریتم *آسانو*[[53]](#footnote-53) و الگوریتم *سوری*[[54]](#footnote-54) با زمان اجرای O(n log n) و الگوریتم بهینه *دهنی*[[55]](#footnote-55) با زمان اجرای O(n+h log h) اشاره کرد [14،13،12]. *زارعی* و *قدسی* نیز الگوریتمی برای نسخه بر مبنای پرس و جوی ­این مسئله[[56]](#footnote-56) ارائه کردند [15].

### 2-1-5 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 2-10 متوجه شد.

1

2

3

4

5

6

7

8

9

10

11

12

13

L1

L2

راس معمولی

راس شروع

راس پایان

راس تفکیک

راس ادغام

شکل 2-10) یک چندضلعی ساده که یکنواخت قائم نیست.

راس شماره 2 یک راس ادغام است. به دلیل اینکه زاویه داخلی این راس بزرگتر از 180 درجه است و همچنین دو راس مجاور آن (راس­های شماره 1 و 3) 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 8 یک راس تفکیک است. به دلیل اینکه زاویه داخلی آن بزرگتر از 180 درجه است و دو راس مجاور آن (راس­های شماره 7 و 9) 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 1-1 ثابت شده است.

**قضیه 1-1: قضیه چندضلعی یکنواخت قائم**  
اگر یک چندضلعی ساده، راس ادغام یا تفکیکی نداشته باشد آنگاه آن چندضلعی، یکنواخت قائم است.

**اثبات[[57]](#footnote-57)**:  
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 2-11 (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.

L

m

n

r

A

B

**P**

s

راس تفکیک

(الف)

L

m=r

n

r’

**P**

s

B

A

راس ادغام

(ب)

شکل 2-11) 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≠m باشد (شکل 2-11 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.

اگر r=m باشد (شکل 2-11 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 2-1 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 2-12 یک مثال (نقض) است که این ادعا را ثابت می­کند. این چندضلعی یکنواخت قائم است زیرا اگر راس­های چندضلعی از بالاترین راس (راس شماره 1) به سمت پایین­ترین راس (راس شماره 3) 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 6 یک راس ادغام است.

1

2

3

4

5

6

شکل 2-12) یک مثال برای نشان دادن اینکه قضیه 1-1 در جهت عکس برقرار نیست.

## 2-2 الگوریتم لی و پریپاراتا

قضیه 2-1 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[[58]](#footnote-58) یک الگوریتم با زمان اجرای O(n log n) برای حل مسئله تجزیه یک چندضلعی ساده به چندضلعی­های قائم ارائه کردند [3]. 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 2-13 مراحل تجزیه یک چندضلعی ساده که با این الگوریتم تجزیه شده است را نشان می­دهد.

(الف)

1

2

3

4

5

6

7

8

9

(ب)

1

2

3

4

5

6

7

8

9

P1

P2

(پ)

1

2

3

4

5

6

7

8

9

P1

P2

P3

شکل 2-13) مراحل تجزیه یک چندضلعی ساده توسط الگوریتم *لی* و *پریپاراتا*

شکل 2-13 (الف) یک چندضلعی ساده را نمایش می­دهد. راس شماره 2 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 5، 6، 7 و 8 گزینه­های مناسبی هستند. در اینجا، راس شماره 5 انتخاب شده است. بعد از اینکه راس­های شماره 2 و 5 با یک قطر به هم متصل شدند، چندضلعی اولیه به دو چندضلعی P1 و P2 تجزیه می­شود.  
راس شماره 2 (که در چندضلعی اولیه یک راس ادغام است) در چندضلعی­های P1 و P2 یک راس معمولی است. راس شماره 7 یک راس تفکیک است و باید توسط یک قطر به راس مناسبی که در بالای آن قرار گرفته متصل شود. راس­های شماره 1، 2، 5 و 9 گزینه­های مناسبی هستند. در اینجا راس شماره 5 انتخاب شده است. با اضافه کردن قطر دوم، چندضلعی اولیه به سه چندضلعی P1، P2 و P3 تجزیه می­شود که هیچکدام راس ادغام یا تفکیک ندارند. بنابراین با توجه به قضیه 1-1 این چندضلعی­ها یکنواخت قائم هستند.

برای اضافه کردن قطر به راس­های ادغام و تفکیک، از روش جاروب صفحه[[59]](#footnote-59) استفاده می­شود.  
فرض کنید که v1، v2، ... ، vn راس­های یک چندضلعی ساده به نام P باشد که به صورت پاد ساعتگرد نامگذاری شده است. همچنین فرض کنید 1e، 2e، ... ، en مجموعه اضلاع این چندضلعی باشد به طوری که برای ≤i<n1، و . در روش جاروب صفحه، یک خط جاروب[[60]](#footnote-60) فرضی به نام L صفحه را از بالا به سمت پایین پویش[[61]](#footnote-61) می­کند. خط جاروب در نقاط خاصی متوقف می­شود.  
به این نقاط، نقاط پردازشی[[62]](#footnote-62) 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[[63]](#footnote-63) به نام Q ذخیره می­شوند. اولویت نقاط داخل صف با توجه به مقدار 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 vi می­رسد. یک سوال مهم این است که راس vi به چه راسی متصل شود؟ یک گزینه مناسب، راسی است که نزدیک vi 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 ej ضلعی است که (بلافاصله) در سمت چپ راس vi قرار دارد  
(خط جاروب ضلع ej را قطع می­کند). این وضعیت در شکل 2-14 نمایش داده شده است.  
همواره می­توان راس vi را به پایین­ترین راسی که بین ضلع­های ei و ek قرار گرفته و بالاتر از vi است متصل نمود. اگر چنین راسی وجود نداشته باشد، می­توان vi را به راس بالایی ضلع ej و یا ek متصل کرد. چنین راسی را نقطه کمکی ej می­نامیم و آن را با helper(ej) نشان می­دهیم. نقطه کمکی ej پایین­ترین راسی است که بالای خط جاروب قرار دارد به طوری که پاره­خط افقی که این راس را به ضلع ej متصل می­کند تماما درون چندضلعی P واقع می­شود. باید توجه داشت که نقطه کمکی ej می­تواند راس بالایی ej باشد.

L

ej

ek

helper(ej)

ei

ei-1

vi

شکل 2-14) 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 vi می­رسد. همچنین فرض کنید ej و ek اضلاعی باشند که در سمت چپ و راست vi واقع شده­اند. این وضعیت در شکل 2-15 نمایش داده شده است.

ej

vi

L

vm

ek

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

شکل 2-15) پردازش یک راس ادغام

ملاحظه می­شود که وقتی خط جاروب به این راس می­رسد، این راس تبدیل به راس کمکی جدیدی برای ضلع ej می­شود. هدف این است که راس vi به بالاترین راسی که در زیر خط جاروب و بین ضلع­های ej و ek 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 180 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 vi 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 vm می­رسد و این راس جایگزین راس vi (به عنوان راس کمکی ضلع ej) 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 ej با راس دیگری جایگزین نشود. در این حالت راس vi به راس پایینی ضلع ej 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[[64]](#footnote-64) 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.

در ادامه، پیاده­سازی این الگوریتم با استفاده از شش رویه[[65]](#footnote-65) ملاحظه می­شود[[66]](#footnote-66). 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 vi with the highest priority from Q

5. Call the appropriate procedure to handle the vertex, depending on its type.

الگوریتم 2-1) الگوریتم تجزیه چندضلعی با الگوریتم لی و پریپاراتا

**HandleStartVertex**(vi)

1. Insert ei in T and set helper(ei) to vi.

ادامه الگوریتم 2-1) پردازش راس­های شروع

**HandleEndVertex**(vi)

1. **if** helper(ei-1) is a merge vertex **then**

2. Insert the diagonal connecting vi to helper (ei-1)

3. Delete ei-1 from T

ادامه الگوریتم 2-1) پردازش راس­های پایان

**HandleSplitVertex**(vi)

1. Search in T to find the edge ej directly left of vi.

2. Insert the diagonal connecting vi to helper (ej) in D.

3. helper(ej) ← vi

4. Insert ei in T and set helper(ei) to vi.

ادامه الگوریتم 2-1) پردازش راس­های تفکیک

**HandleMergeVertex**(vi)

1. **if** helper(ej-1) is a merge vertex

2. **then** Insert the diagonal connectiong vi to helper(ei-1) in D

3. Delete ei-1 from T.

4. Search in T to find the edge ej directly left of vi.

5. **if** helper(ej) is a merge vertex

6. **then** Insert the diagonal connecting vi to helper(ej) in D.

7. helper(ej) ← vi

ادامه الگوریتم 2-1) پردازش راس­های ادغام

**HandleRegularVertex**(vi)

1. **if** the interior of P lies to the right of vi

2. **then if** helper(ej-1) is a merge vertex

3. **then** Insert the diagonal connecting vi to helper(ei-1) in D.

4. Delete ei-1 from T.

5. Insert ei in T and set helper(ei) to vi.

6. **else** Search in T to find the edge ej directly left of vi.

7. **if** helper(ej) is a merge vertex

8. **then** Insert the diagonal connecting vi to helper(ej) in D.

9. helper(ej) ← vi

ادامه الگوریتم 2-1) پردازش راس­های معمولی

## 2-3 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 [16]. لیو و نتافوس یک الگوریتم از مرتبه  
O(nN3 + N2n log n + N5) برای تجزیه کمینه یک چندضلعی ساده به چندضلعی­های یکنواخت ارائه کرده­اند [5]. 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[[67]](#footnote-67) است. یک مجموعه مستقل[[68]](#footnote-68) در یک گراف بدون جهت، زیرمجموعه­ای از راس­های 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[[69]](#footnote-69) گفته می­شود. مجموعه مستقل بیشینه یک گراف الزاما یکتا نیست.

1

2

3

4

5

6

**S1** = {1, 4}  
**S2** = {1, 3, 6}

**S3** = {1, 4, 5}

**S4** = {4, 5}

شکل 2-16) یک گراف و چهار مجموعه مستقل آن

شکل 2-16 یک گراف و چهار مجموعه مستقل آن را نشان می­دهد. مجموعه­های S2 و S3 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 [[70]](#footnote-70) است [24]. با این حال الگوریتم­های با پیچیدگی زمانی چندجمله­ای برای یافتن مجموعه مستقل بیشینه یک گراف مدور وجود دارد [23،22،21،20،19،18،17،5].

گراف تقاطع[[71]](#footnote-71) قطرهای یک دایره مفروض به نام C، گراف مدور[[72]](#footnote-72) 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 [24]. تا به حال الگوریتم‏های متعددی برای یافتن مجموعه مستقل بیشینه یک گراف مدور ارائه شده است [15،14،13،12،11،10،9،4]. *لیو* و *نتافوس* در کنار الگوریتم پیشنهادی خود برای تجزیه کمینه چندضلعی، یک الگوریتم از مرتبه O(n3) برای یافتن یک مجموعه مستقل بیشینه یک گراف مدور با n راس ارائه کردند [4].

**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[[73]](#footnote-73) است متصل شود. همچنین برای 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[[74]](#footnote-74) 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[[75]](#footnote-75)) 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.

**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 [25].

**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[[76]](#footnote-76) 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(N3) است [5]. گام چهارم در زمان O(n log n) توسط الگوریتم تجزیه یکنواخت *لی* و *پریپاراتا* اجرا می­شود [25]. بنابراین این الگوریتم یک چندضلعی ساده را در زمان O(n log n + nN + N3) 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(N2) خط است [5]. بنابراین الگوریتم *لیو* و *نتافوس* یک چندضلعی ساده را در زمان O(nN2 log n + nN3 + N5) به صورت کمینه (نسبت به هر خطی) به چندضلعی­های یکنواخت تجزیه می­کند.

## 2-4 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 1991 *چازل* یک الگوریتم خطی (کاملا پیچیده) برای حل این مسئله ارائه کرد [26]. پیچیدگی پیاده­سازی این الگوریتم باعث شده است که در عمل از الگوریتم­های تقریبا خطی استفاده شود[[77]](#footnote-77). بعد از ارائه این الگوریتم توسط *چازل*، تلاش برای یافتن الگوریتم­های ساده­تر متوقف نشد. در سال 1994 *سایدل*[[78]](#footnote-78) یک الگوریتم تصادفی افزایشی[[79]](#footnote-79) برای مثلثی‏سازی یک چندضلعی ارائه کرد [7]. 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 ≥ 1 .

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

**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 [1] 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.

### 2-4-1 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 2-17 (الف) یک چندضلعی که به صورت افقی (تجزیه) ذوزنقه­ای شده است ملاحظه می­شود.  
این چندضلعی به 9 ذوزنقه تجزیه شده است.

**(الف)**

**(ب)**

**(ج)**

شکل 2-17) 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 2-17 (ج) ملاحظه می­شود.

## 2-4 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:

1. پیچیدگی زمان و فضای الگوریتم
2. تعداد چندضلعی­های حاصل از تجزیه
3. ایجاد و یا عدم ایجاد نقاط کمکی[[80]](#footnote-80)
4. توانایی تجزیه چندضلعی­های حفره‌دار

شاخص اول برای ارزیابی کارآیی هر الگوریتمی مورد بررسی قرار می­گیرد. در بسیاری از کاربردهای هندسه محاسباتی مانند گرافیک برداری[[81]](#footnote-81) و چندضلعی­های مورد استفاده در سامانه­های اطلاعات جغرافیایی[[82]](#footnote-82) تعداد راس­های چندضلعی و یا تعداد کل چندضلعی­ها به چندین هزار می­رسد [8]. پردازش این چندضلعی­ها در یک زمان قابل قبول، مخصوصا در کاربردهای بلادرنگ[[83]](#footnote-83) مانند بازی­های رایانه­ای که هر صحنه باید به سرعت ارائه[[84]](#footnote-84) 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[[85]](#footnote-85) و نویسه ­خوانی نوری[[86]](#footnote-86) 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[[87]](#footnote-87) 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 الگوریتم قادر باشد چندضلعی­های حفره‌دار را نیز تجزیه کند، یک مزیت محسوب می­شود زیرا در عمل با چنین چندضلعی­هایی مواجه می­شویم.

در جدول شماره 2-1 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 180 درجه است را نشان می­دهد.

جدول 2-1) 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* [1] | T(n) ∈ O(n log n)  S(n) ∈ O(n) | خیر | خیر | خیر |
| *لیو* و *نتافوس* [5] | T(n) ∈ O(nN3 + N2n log n + N5)  S(n) ∈ O(n2) | بله | خیر | خیر |
| *لیو* و *نتافوس* [5] | T(n) ∈ O(n N3 log n + N5)  S(n) ∈ O(n2) | بله | بله | خیر |
| *سایدل* [7] | ERT[[88]](#footnote-88) ∈ O(n log\* n)  EMU[[89]](#footnote-89) ∈ O(n) | خیر | خیر | بله[[90]](#footnote-90) |
| *وی* [27] | T(n) ∈ O(K(n log n + h log3 h))  S(n) ∈ O(n) | بله | بله | بله |
| *کیل* [4] | T(n) ∈ O(n4N) | بله[[91]](#footnote-91) | 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 صورت کمینه انجام می­دهد اما زمان اجرای آن مطلوب نیست. با توجه اینکه در بدترین حالت N از مرتبه O(n) می­باشد[[92]](#footnote-92)، زمان اجرای بدترین حالت این الگوریتم از مرتبه O(n5) 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 1983 کیل ثابت کرد که مسئله تجزیه کمینه یک چندضلعی حفره‌دار به چندضلعی­های یکنواخت، بدون استفاده از نقاط کمکی، یک مسئله NP-سخت است [4].

# 

# فصل 3 ارائه یک الگوریتم حریصانه جهت تجزیه یکنواخت چندضلعی

در این فصل یک الگوریتم حریصانه[[93]](#footnote-93) 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 3-1 ابتدا ایده اصلی این الگوریتم در قالب یک الگوریتم اولیه مطرح می­شود، سپس در بخش 3-2 زمان اجرای الگوریتم اولیه با استفاده از روش­هایی بهبود داده­ می­شود. در بخش 3-3 نیز کارآیی الگوریتم جدید با الگوریتم *لی* و الگوریتم *سایدل* مقایسه می­شود.

## 3-1 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.

### 3-1-2 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  
3-1 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 3-1) الگوریتم حریصانه اولیه

**تحلیل پیچیدگی زمانی الگوریتم 3-1**

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 Nm است.

در گام 2-2 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 2-2 به زمان O(n) نیاز دارد. بنابراین زمان اجرای گام دوم برابر است با:

(3-1) t2(n, Nm) = Nm (O(n) + O(n)) ∈ O(n×Nm)

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 Ns باشد. بنابراین زمان اجرای بدترین حالت گام سوم برابر است با:

(3-2) t3(n, Ns) = Ns × O(n) ∈ O(n×Ns)

در نتیجه زمان اجرای الگوریتم 3-1 برابر مقدار زیر است:

(3-3) t(n, Nm, Ns) = O(n) + O(n×Nm) + O(n×Ns) ∈ O(n) + O(n (Nm+Ns))

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

(3-4) t(n) ∈ O(n) + O(n×N) ∈ O(n + n×N)

## 3-2 بهبود الگوریتم اولیه

زمان اجرای الگوریتم 3-1 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 3-1 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.

### 3-2-1 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 3-1 یک چندضلعی که به صورت ذوزنقه­ای تجزیه شده است ملاحظه می­شود. این چندضلعی به 10 ذوزنقه تجزیه شده است (مثلث‏ها نیز ذوزنقه­هایی فرض شده­اند که طول یکی از قاعده­های آنها صفر است).

1

2

3

4

5

6

7

8

9

10

11

a

b

c

d

e

f

g

h

i

j

شکل 3-1) 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[[94]](#footnote-94) در حافظه وجود دارد که حاوی اطلاعات زیر است:

1. مختصات هندسی چهار راس ذوزنقه

2. 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 1 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 1 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[[95]](#footnote-95) باشد. در هنگام ملاقات هر ذوزنقه، بخشی از قاعده پایینی آن که از راس ادغام، آشکار[[96]](#footnote-96) است محاسبه می­شود. شکل 3-2 ناحیه آشکار از راس شماره 1روی قاعده پایینی ذوزنقه­های c، d، e و f را نشان می­دهد.

1

c

1

d

1

e

1

f

(الف)

(ب)

(پ)

(ت)

شکل 3-2) ناحیه آشکار از راس شماره 1روی 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[[97]](#footnote-97) 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.

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 [1].  
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 3-3 دو چندضلعی با تعداد راس­های برابر که به صورت ذوزنقه­ای تجزیه شده­اند رسم شده است.  
در چندضلعی (الف) مقدار K باید حداقل برابر 10 باشد اما برای چندضلعی (ب) مقدار 3=K کفایت می­کند.

(الف)

(ب)

شکل 3-3) دو چندضلعی ساده با 15 راس

در الگوریتم 3-2، 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

1. Found ← **false**, i ← 0

2. t ← AdjacentTrapezoid(v)

3. **while** ((**not** Found) **and** (i < K) **and** (t ≠ **null**))

4. VisiblePortion ← ComputeVisiblePortion(t)

5. **if** (Length(VisiblePortion) == 0)

6. **exit** **while**

7. **if** (t has only 1 lower adjacent trapezoid)

8. t ← GetLowerTrapezoid(t)

9. **else if** (t has 2 lower adjacent trapezoids)

10. SplitVertex ← GetSplitVertex(t)

11. **if** (VisiblePortion.X1 ≤ SplitVertex.X ≤ VisiblePortion.X2)

12. Found ← **true**

13. AddDiagonal (v, SplitVertex)

14. **else if** (VisiblePortion.X1 < SplitVertex.X)

15. t ← BottomLeftTrapezoid(t)

16. **else**

17. t ← BottomRightTrapezoid(t)

18. i ← i + 1

19. **if** (**not** Found)

20. AddDiagonal(v, BottomVertex(AdjacentTrapezoid(v)))

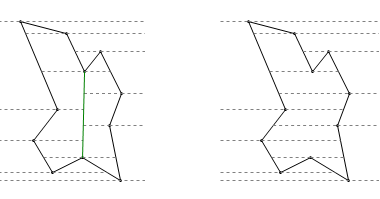
21. **end**.

الگوریتم 3-2) 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 3-4 یک چندضلعی را قبل و بعد از اجرای رویه HandleMergeVertex نشان می­دهد.

(الف)

(ب)



شکل 3-4) قبل و بعد از اضافه شدن یک قطر به چندضلعی

برای اتصال یک راس تفکیک به یک راس ادغام می­توان مشابه با الگوریتم 3-2 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.

### 3-2-2 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 [9،8،7]. 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 3-1 دارای دو مزیت است:

1. 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 20 و یا 30 کفایت می­کند.

2. قابلیت تجزیه چندضلعی­های حفره­دار

الگوریتم 3-1 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 3-3) 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 Nm و تعداد راس­های تفکیک چندضلعی برابر Ns باشد، زمان اجرای گام چهارم از مرتبه O(K×Nm) و گام پنجم از مرتبه O (Ns) خواهد بود. بنابراین زمان کل اجرای الگوریتم 3-3 از رابطه زیر بدست می­آید:

(3-5) t(n, Nm, Ns) ∈ O(T) + O(n) + O(K×Nm) + O(Ns)

با توجه به اینکه Ns ∈ O(n) و همچنین تجزیه ذوزنقه­ای یک چندضلعی با n راس از مرتبه Ω(n) است، رابطه فوق را می­توان به صورت زیر بازنویسی کرد:

(3-6) t(n, Nm) ∈ O(T) + O(K×Nm)

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

(3-7) t(n, Nm) ∈ O(T) + O(Nm) t(n, Nm) ∈ O(T)

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∈O(n) باشد، آنگاه هزینه کلی الگوریتم 3-3 از مرتبه زیر است:

(3-9) t(n, Nm) ∈ O(T) + O(n×Nm)

## 3-3 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[[98]](#footnote-98) الگوریتم تصادفی افزایشی سایدل استفاده کرده­ایم [7]. 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 1000 راس دارند با 10K= نتایج مطلوبی حاصل شده است).

در جدول 3-1 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 1600 چندضلعی ساده که به صورت تصادفی تولید شده­اند استفاده شده است[[99]](#footnote-99). تعداد راس­های چندضلعی­های تولید شده در ستون اول جدول 3-1 ملاحظه می­شود. به ازای هر مقدار مانند v که در این ستون درج شده است، یک مجموعه با 100 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 100 چندضلعی، تقسیم بر عدد 100).

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.

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

جدول 3-1) مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی*

برای مقایسه آسان­تر این سه الگوریتم، داده­های جدول 3-1 در نمودار 3-1 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 5، 10، 20 و 30 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 3-1) مقایسه الگوریتم پیشنهادی با الگوریتم *سایدل* و الگوریتم *لی*

# فصل 4

# 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 4-2 پیشنهاداتی برای بهبود عملکرد این الگوریتم ارائه شده است.

## 4-1 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 3-3 نتایج حاصل از پیاده­سازی الگوریتم پیشنهادی و مقایسه آن با الگوریتم لی و سایدل بحث و بررسی شده است.

## 4-2 نتیجه­گیری

تا کنون پنج الگوریتم برای تجزیه یکنواخت چندضلعی­های ساده و یا حفره­دار وجود دارد [27].  
جزئیات مربوط به این الگوریتم­ها در جدول 3-1 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(Nn4) و الگوریتم لیو از مرتبه  
O(nN3 + N2n log n + N5) 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 [4].

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 70 درصدی در استفاده از قطرها برای تجزیه چندضلعی­هایی با 1500 راس).

این الگوریتم نسبت به الگوریتم لی نیز از تعداد قطرهای کمتری برای تجزیه چندضلعی استفاده می­کند (به طور میانگین کاهش 17 درصدی در استفاده از قطرها برای تجزیه چندضلعی­هایی با 1500 راس).

## 4-2 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.

### 4-2-1 انتخاب بین تجزیه ذوزنقه­ای افقی یا عمودی

در گام اول الگوریتم 3-3 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 4-1 چنین وضعیتی را نشان می­دهد.

(الف)

(ب)

شکل 4-1) تجزیه ذوزنقه‌ای افقی و عمودی یک چندضلعی ساده

برای دست یافتن به یک قطر ویژه در چندضلعی شکل 4-1(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[[100]](#footnote-100) 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.

### 4-2-2 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 3-1 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.

# 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. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Firstname: Ahmad | | | Surname: Tajedini | | | |
| Thesis Title: An Algorithm for Monotone Decomposition of Polygons in 2D Space | | | | | | |
| Supervisor: Dr. J. Karimpour  Advisor: Dr. Sh. Lotfi | | | | | | |
| University: Tabriz | | Field: Computer Systems | | Major: Computer Science | | Degree: MSc |
| Pages: 65 | Graduation Date: September 11, 2013 | | | | Faculty: Mathematical Science | |
| 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

1. polygon decomposition [↑](#footnote-ref-1)
2. convex [↑](#footnote-ref-2)
3. monotone with respect to a line [↑](#footnote-ref-3)
4. robot motion planning [↑](#footnote-ref-4)
5. Steiner points [↑](#footnote-ref-5)
6. monotone decomposition [↑](#footnote-ref-6)
7. uniformly monotone with respect to axis Y [↑](#footnote-ref-7)
8. Minimum Monotone Decomposition [↑](#footnote-ref-8)
9. simple [↑](#footnote-ref-9)
10. uniform [↑](#footnote-ref-10)
11. convex [↑](#footnote-ref-11)
12. star-shaped [↑](#footnote-ref-12)
13. trapezoidal [↑](#footnote-ref-13)
14. minimum ink decomposition [↑](#footnote-ref-14)
15. reflex vertex [↑](#footnote-ref-15)
16. notch vertex [↑](#footnote-ref-16)
17. hole [↑](#footnote-ref-17)
18. steiner points [↑](#footnote-ref-18)
19. Lee and Preparata [↑](#footnote-ref-19)
20. M. Keil [↑](#footnote-ref-20)
21. Wei [↑](#footnote-ref-21)
22. visibility graph [↑](#footnote-ref-22)
23. Lee and Preparata [↑](#footnote-ref-23)
24. 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. [↑](#footnote-ref-24)
25. triangulation [↑](#footnote-ref-25)
26. monotone with respect line L [↑](#footnote-ref-26)
27. uniformly monotone with respect to line L [↑](#footnote-ref-27)
28. Y-monotone polygon [↑](#footnote-ref-28)
29. 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. [↑](#footnote-ref-29)
30. turn vertex [↑](#footnote-ref-30)
31. regular Vertex [↑](#footnote-ref-31)
32. start Vertex [↑](#footnote-ref-32)
33. end Vertex [↑](#footnote-ref-33)
34. merge Vertex [↑](#footnote-ref-34)
35. split Vertex [↑](#footnote-ref-35)
36. data structures [↑](#footnote-ref-36)
37. linked List [↑](#footnote-ref-37)
38. subspace [↑](#footnote-ref-38)
39. trapezoidal decomposition [↑](#footnote-ref-39)
40. 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. [↑](#footnote-ref-40)
41. Chazelle & Incerpi [↑](#footnote-ref-41)
42. robot motion planning [↑](#footnote-ref-42)
43. point location problem [↑](#footnote-ref-43)
44. Seidel [↑](#footnote-ref-44)
45. incremental randomized algorithm [↑](#footnote-ref-45)
46. expected running time [↑](#footnote-ref-46)
47. expected memory usage [↑](#footnote-ref-47)
48. یا نقطه a از نقطه b آشکار است (a is visible from b) و برعکس. [↑](#footnote-ref-48)
49. visibility polygon from point a [↑](#footnote-ref-49)
50. visibility polygon from an edge [↑](#footnote-ref-50)
51. H. El Gindy [↑](#footnote-ref-51)
52. D. T. Lee [↑](#footnote-ref-52)
53. T. Asano [↑](#footnote-ref-53)
54. S. Suri [↑](#footnote-ref-54)
55. F. Dehne [↑](#footnote-ref-55)
56. query based version of the (visibility) problem [↑](#footnote-ref-56)
57. اثبات این قضیه از مرجع [1] اخذ شده است. [↑](#footnote-ref-57)
58. Lee and Preparata [↑](#footnote-ref-58)
59. plane sweep method [↑](#footnote-ref-59)
60. sweep line [↑](#footnote-ref-60)
61. scan [↑](#footnote-ref-61)
62. event points [↑](#footnote-ref-62)
63. priority queue [↑](#footnote-ref-63)
64. binary tree [↑](#footnote-ref-64)
65. procedure [↑](#footnote-ref-65)
66. شبه کد مربوط به پیاده­سازی این الگوریتم از مرجع [1] اخذ شده است. [↑](#footnote-ref-66)
67. circle graph [↑](#footnote-ref-67)
68. independent set [↑](#footnote-ref-68)
69. maximum independent set [↑](#footnote-ref-69)
70. NP-Hard [↑](#footnote-ref-70)
71. intersection graph [↑](#footnote-ref-71)
72. circle graph [↑](#footnote-ref-72)
73. Visible [↑](#footnote-ref-73)
74. turn vertex [↑](#footnote-ref-74)
75. visible to each other [↑](#footnote-ref-75)
76. visibility polygon [↑](#footnote-ref-76)
77. 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. [↑](#footnote-ref-77)
78. Seidel [↑](#footnote-ref-78)
79. incremental randomized algorithm [↑](#footnote-ref-79)
80. steiner points [↑](#footnote-ref-80)
81. vector graphic [↑](#footnote-ref-81)
82. Geographical Information Systems (GIS) [↑](#footnote-ref-82)
83. real time [↑](#footnote-ref-83)
84. render [↑](#footnote-ref-84)
85. machine vision [↑](#footnote-ref-85)
86. Optical Character Recognition (OCR) [↑](#footnote-ref-86)
87. visual [↑](#footnote-ref-87)
88. Expected Running Time (ERT) [↑](#footnote-ref-88)
89. Expected Memory Usage (EMU) [↑](#footnote-ref-89)
90. در مقاله *سایدل* به این موضوع اشاره­ای نشده است اما با اندکی ویرایش این الگوریتم می­توان چندضلعی­های  
    حفره­دار را نیز تجزیه کرد [22]. [↑](#footnote-ref-90)
91. تجزیه به صورت کمینه انجام می­گیرد اما الزاما یکپارچه یکنواخت (uniformly monotone) نیست [4]. [↑](#footnote-ref-91)
92. مقدار N وابسته به n است. در حالت کلی، هرچه مقدار n بیشتر می­شود، مقدار N نیز به صورت خطی افزایش پیدا می­کند. [↑](#footnote-ref-92)
93. greedy algorithm [↑](#footnote-ref-93)
94. node [↑](#footnote-ref-94)
95. visible [↑](#footnote-ref-95)
96. Visible [↑](#footnote-ref-96)
97. Incremental [↑](#footnote-ref-97)
98. 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 [22]. [↑](#footnote-ref-98)
99. چندضلعی­های تصادفی را با استفاده از کتابخانه CGAL تولید کردیم. [↑](#footnote-ref-99)
100. preprocess [↑](#footnote-ref-100)