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

تجزیه چندضلعی به اجزای تقریبا محدب با الگوریتم IFACD

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

[۱] گروه علوم کامپیوتر، دانشکده علوم ریاضی، دانشگاه تبریز، تبریز،
mehdi.byt@gmail.com

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

[۳] گروه علوم کامپیوتر، دانشکده علوم ریاضی، دانشگاه تبریز، تبریز،
mehrta.cs@gmail.com

چکیده:

مسئله تجزیه چندضلعی‌ها یک مسئله کلاسیک در هندسه محاسباتی است، که همواره از بحث‌های مورد علاقه پژوهش‌گران بوده است. اجزای تولید شده از تجزیه چندضلعی به اجزای تقریبا محدب نسبت به اجزای تولید شده محدب، قابلیت محاسباتی بالاتری دارند و از نظر تعداد قابل مدیریت‌ترند. یک روش جدید برای تجزیه چندضلعی ساده به اجزای تقریبا محدب الگوریتم FACD[۱] است، در این الگوریتم کیفیت اجزای تولید شده بهبود یافته و تجزیه‌ای با میزان بصری بودن بالا تولید می‌شود. یکی از معایب این الگوریتم پیچیدگی زمانی نسبتا بالای آن می‌باشد. از آن‌جا که در مباحث گرافیک کامپیوتری زمان محاسبات یک فاکتور مهم به حساب می‌آید، ما در این پژوهش پیچیدگی زمانی FACD را محاسبه کرده و با انجام یک پیش‌پردازش پیچیدگی زمانی آن را کاهش داده‌ایم و الگوریتم بهبود یافته را با نام الگوریتم IFACD[۲] ارائه کرده‌ایم.

کلمات کلیدی:

برنامه نویسی پویا، پوسته محدب، تقعر نسبی، چندضلعی محدب، گراف برش، راس پاکت کمینه، مثلث‌بندی دلونی.

۱- مقدمه

تجزیه اشکال یک بخش مهم در آنالیز اشیا[۳] و ادراک[۴] می‌باشد [6]. مسائل کاربردی زیادی در زمینه بینایی ماشین و گرافیک کامپیوتری مانند ساده سازی اشکال[۵]، کشف برخورد و استخراج اسکلت[۶] از یک تجزیه کارآمد و قابل اطمینان بهره می‌برند [7,8,4]. مبحث تجزیه شکل ها به دو قسمت تجزیه به شکل‌های با معنی[۷] و دیگری تجزیه به شکل های هندسی تقسیم می‌شود [9]. منظور از قطعات بامعنی، بخش‌هایی است که توسط قوه ادراک انسان راحت‌تر تجزیه و تحلیل می‌شوند. بنابراین برای آن‌ها تعریف دقیقی وجود ندارد. تجزیه اشکال پیچیده دوبعدی و سه‌بعدی به اشکال کوچک‌تر و قابل محاسبه‌تر و قابل مدیریت‌تر می‌تواند پردازش آن‌ها را برای کامپیوتر در برنامه‌های کاربردی (مثل برنامه‌های گرافیکی) و در بینایی ماشین‌ها آسان‌تر کند.

در میان انواع تجزیه، به دو دلیل تجزیه به اشکال محدب از اهمیت ویژه‌ای نسبت به دیگر انواع تجزیه برخوردار است: (۱) در ریاضیات قضایای فراوانی برای چندضلعی‌های محدب وجود دارد که در مواقع لزوم می‌توان از آن‌ها بهره برد، (۲) الگوریتم‌های بسیاری وجود دارد که بر روی چندضلعی‌های محدب بسیار سریع‌تر و کارآمدتر اجرا می‌شوند [3]. مسئله تجزیه چندضلعی می‌تواند به گروه‌های تجزیه چندضلعی باحفره، بدون حفره، ساده، غیرساده، دوبعدی، سه بعدی و غیره تقسیم‌بندی شود. در این مقاله موضوع بحث ما تجزیه چندضلعی‌های ساده بدون حفره در فضای دو بعدی است.

اگر در تجزیه‌ای نیاز باشد که تعداد اجزای تولید شده کمینه شود، مسئله تجزیه چندضلعی به اشکال محدب می‌تواند موضوع مسائل بهینه سازی واقع شود. برای چندضلعی‌های با حفره، این مسئله در حالت کمینه np-hard است [4]. اگر مسئله تعداد اجزای کمینه را برای چندضلعی‌های بدون حفره در نظر بگیریم، مسئله به این موضوع که آیا مجاز به استفاده از نقاط اضافی (اضافه کردن راس به چندضلعی) هستیم یا خیر وابسته می‌شود. هنگامی که جواب این سوال منفی باشد یعنی اجازه استفاده از نقاط اضافی را به عنوان رئوس کمکی نداشته باشیم Chazelle [10] یک الگوریتم با مرتبه زمانی $O(n \log n)$ ارائه کرده است (n تعداد رئوس چندضلعی). بعد از این الگوریتم، Greene [11] الگوریتمی با مرتبه زمانی $O(n^2 r^2)$ ارائه دادکه حالت بهینه تعداد اجزا را تولید می‌کرد (r تعداد رئوس بازگشتی است و یک راس بازگشتی راسی است که زاویه داخلی آن بیش از ۱۸۰ درجه باشد [5]). Keil [12] توانست زمان الگوریتم ارائه شده در [11] را به زمان قابل قبول‌تر

$O(r^2 n \log n)$ بهبود بخشد و سپس به همراه Snoeyink زمان این الگوریتم را به $O(n + r^2 \min (r^2, n))$ کاهش دادند [14]. هنگامی که مجاز به استفاده از نقاط اضافی باشیم Chazelle و Dobkin [15] الگوریتمی با مرتبه زمانی $O(n + r^3)$ ارائه کرده‌اند.

تجزیه به اشکال محدب غالبا وقت‌گیر است و تعداد اشکال تولید شده می‌تواند تا حدی زیاد باشند که غیر قابل مدیریت شوند. در مثالی که در [4] مطرح شده است، شکلی توسط دو الگوریتم تجزیه به اجزای محدب و تقریبا محدب تجزیه شده است. وقتی این شکل با یک الگوریتم تجزیه به اجزای محدب تجزیه شده است، بیش از ۷۲۶۲۴۰ قطعه تولید شده و ذخیره سازی آن ۲۰ گیگا بایت فضا اشغال کرده است و تجزیه این شکل حدود ۴ ساعت به طول انجامیده در حالی که وقتی همین شکل با الگوریتم تجزیه به اجزای تقریبا محدب تجزیه شده است، تنها ۹۸ قطعه تولید شده و ذخیره سازی آن فضایی حدود ۱۴ مگابایت اشغال کرده و تجزیه آن تنها ۲۳۲ ثانیه طول کشیده است. بنابراین برخی محققان بر آن شدند تا شکل اولیه را به اشکال تقریبا محدب تجزیه کنند تا بدین ترتیب در زمان محاسبات صرفه‌جویی کرده و تعداد اجزای تولید شده را کاهش دهند.

اولین کار انجام شده در زمینه تجزیه به اجزای تقریبا محدب در سال ۲۰۰۶ توسط Lien و Amato انجام گرفت. آن‌ها یک الگوریتم بازگشتی از مرتبه $O(nr)$ (n تعداد رئوس چندضلعی و r تعداد راس‌های بازگشتی می‌باشد) برای تجزیه به اشکال تقریبا محدب در محیط دوبعدی و سه بعدی پیشنهاد کردند [1]. این الگوریتم یک مقدار τ (ضریب تقعر) را به عنوان ورودی مسئله دریافت می‌کند و چندضلعی‌هایی را تولید می‌کند که تقعری کمتر از ضریب تقعر داشته باشند. تجزیه‌ای که الگوریتم بازگشتی آن‌ها انجام می‌داد نسبت به تجزیه به اشکال محدب بسیار سریع‌تر عمل می‌کرد و در کاربردهای فراوانی که نیازی نبود تا چندضلعی به اجزای کاملا محدب تجزیه شود، قدرت خود را نشان می‌داد.

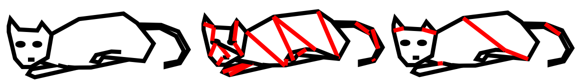
ضعف این الگوریتم در ایجاد یک تجزیه کورکورانه است که بر روی کیفیت اجزای تولید شده هیچ بحثی نمی‌کند و همچنین با تجزیه نویزها و پستی بلندی‌های لبه چندضلعی می‌تواند تجزیه را بی‌هدف جلوه دهد و بخش‌های اضافی زیادی تولید کند و سبب شود اجزایی با میزان بصری بودن[8] پایین حاصل شوند که این کار ممکن است در بعضی از کاربردها اصلا کارآمد نباشد. همانطور که در شکل (۱) مشخص است، از دید یک انسان تجزیه‌ای بامعناتر است که بتواند ناحیه گوش‌ها، صورت، گردن، دست‌ها و دیگر اجزا را جدا کند (این مطلب در تشخیص اشیا بسیار اهمیت دارد) اما الگوریتم آن‌ها هرگز قادر به چنین کاری نبود. از دیگر معایب این الگوریتم آن است که به دلیل استفاده از یک ضریب تقعر ثابت در کل الگوریتم، دو شکل یکسان با مقیاس‌های متفاوت را به دو صورت کاملا متفاوت تجزیه می‌کند که این موضوع می‌تواند در تشخیص اشیا، ماشین را دچار اشتباه کند و یک شکل با دو مقیاس متفاوت را به عنوان دو شکل کاملا متفاوت تشخیص دهد.

Ghosh و همکارانش [3] الگوریتمی با نام FACD ارائه کردند که با استفاده از مفهوم تقعر نسبی تا حد زیادی قادر بود تجزیه‌ای را روی یک شکل انجام دهد که به تجزیه‌ای که یک انسان از آن شکل انجام می‌دهد، نزدیک باشد و حتی‌الامکان معایب الگوریتم‌های قبلی را نداشته باشد، در FACD یک راهکار جدید بر اساس برنامه نویسی پویا[9] برای ارزیابی تمام برش‌های ممکن در پیش گرفته شده است که هدفش کاهش تقعر نسبی نسبت به تقعر مطلق می‌باشد (تعاریف تقعر نسبی و مطلق در [3] آمده است). نتایج بدست آمده از FACD نشان می‌دهد که در اشکالی که دارای اجزای کوچک و مهم هستند (مانند انگشتان دست و پا) اجزای طبیعی‌تر با میزان بصری بودن بالاتری تولید می‌شود بدون آنکه در اجزای بزرگ‌تر بخش‌های اضافی تولید شود یا بخواهد پستی بلندی‌های لبه شکل را در تجزیه دخیل کند.

الگوریتم FACD یکی از بهترین الگوریتم‌ها در زمینه تجزیه به اجزای تقریبا محدب می‌باشد که تاکنون ارائه شده است. این الگوریتم از جنبه‌های زیادی بر الگوریتم‌های مشابه برتری دارد که چند مورد از آن‌ها به این ترتیب است:

(۱) برعکس الگوریتم‌های قبلی که بر اساس معیار ثابت ضریب تقعر عمل می‌کردند، FACD تجزیه را بر اساس مفهوم تقعر نسبی انجام می‌دهد که با توجه به شکل ورودی عمل می‌کند. بنابراین برعکس الگوریتم‌های قبلی نویزهای ریز اما با زاویه زیاد که در لبه شکل قرار دارند را تشخیص داده و در تجزیه دخیل نمی‌کند. (۲) حتی‌الامکان محدب‌ترین قطعات را به عنوان یک بخش جدا می‌کند. (۳) اشکال یکسان با اندازه‌های متفاوت را تقریبا یکسان تجزیه می‌کند (این موضوع در تشخیص اشکال کاربرد دارد). (۴) با کوچک کردن نرخ تقعر برای تجزیه خصوصیات ریز مانند انگشتان دست، قطعات اضافی در بخش‌های دیگر شکل مانند ناحیه کمر یا گردن بوجود نمی‌آید [3].

یکی از معایب این الگوریتم پیچیدگی زمانی نسبتا بالای آن می باشد. ما در اینجا الگوریتم FACD را به چهار گام تجزیه (گام چهارم آن مرحله بازگشتی الگوریتم است) و پیچیدگی زمانی هر گام آن را محاسبه کرده‌ایم (در [3] پیچیدگی زمانی الگوریتم FACD محاسبه نشده است) و در خلال قضایایی اثبات نموده و در اینجا آورده‌ایم. در بخش (۴) پیچیدگی زمانی گام سوم FACD که زمان‌برترین بخش الگوریتم است را از $O(n^4 \log n)$ به زمان $O(n^3 \log n)$ کاهش داده و الگوریتم بهبود داده شده IFACD را ارائه کرده‌ایم.



شکل (۱) : تصویر گربه و دو تجزیه از آن با میزان بصری بودن متفاوت

۲- تجزیه به اجزای تقریبا محدب

اگر تجزیه را به نحوی انجام دهیم که اجزای تولید شده کاملا محدب نباشند و مقداری تقعر در چندضلعی را بپذیریم تا بدین ترتیب در وقت و هزینه صرفه‌جویی کنیم، به آن تجزیه، تجزیه به اجزای تقریبا محدب می‌گویند. در [4] مثالی ذکر شده است که در آن موقعیت 10^8 نقطه را نسبت به یک چندضلعی با استفاده از دو الگوریتم ECD[10] و ACD[11] مشخص کرده است (بررسی کرده که هر نقطه درون چندضلعی قرار می‌گیرد یا خارج از آن). در حالتی که چندضلعی را با الگوریتم ECD تجزیه کرده و سپس موقعیت نقاط را نسبت به چندضلعی مشخص کرده است، چندضلعی به ۵۳۱۹ قطعه محدب تقسیم شده است و یافتن موقعیت نقاط ۵۷ ساعت طول کشیده است. در حالی که وقتی با استفاده از الگوریتم ACD تجزیه را انجام داده، ۲۰۴ قطعه تقریبا محدب حاصل شده که یافتن موقعیت نقاط نسببت به چندضلعی در این حالت تنها حدود ۵۲ دقیقه طول کشیده است و تعداد نقاطی را که الگوریتم اشتباه تشخیص داده است از ۱۰ کمتر است که در مقابل عدد 10^8 می‌توان از آن چشم‌پوشی کرد.

برای چندضلعی P فرض کنید ∂p نشان‌دهنده مجموعه اضلاع و ∂CH_p نشان دهنده پوسته محدب چندضلعی P باشد.

تعریف (۱) : به یال‌هایی از پوسته محدب که یال چندضلعی نباشند و دو راس از چندضلعی را به هم متصل می‌کنند پل می‌گویند (شکل (۲)) [4,1]. به عبارت دیگر

$$\text{Bridges}\ (P) = \partial CH_p \setminus \partial p \qquad (۱)$$

تعریف (۲) : به تعدادی از یال‌های متصل به یکدیگر که یک حلقه بسته را بوجود نیاورند، یا به عبارت دیگر تشکیل چندضلعی ندهند، یک زنجیر[12] می‌گویند [5].

تعریف (۳) : به یال‌هایی از چندضلعی که تشکیل زنجیرهای بیشینه می‌دهند و روی پوسته محدب قرار نداشته باشند، پاکت می‌گویند (شکل (۲)) [4,1]. به عبارت دیگر

$$\text{Pockets}\ (p) = \partial p \setminus \partial CH_p \qquad (۲)$$

تعریف (۴) : فاصله اقلیدسی یک راس بازگشتی درون یک پاکت تا پل مربوط به آن پاکت را تقعر آن راس بازگشتی به روش خط مستقیم می‌گویند [4].

تعریف (۵) : به بیشینه تقعر بین تمام راس‌های یک چندضلعی، تقعر آن چندضلعی می‌گویند [4,1]. به عبارت دیگر

$$\text{Concavity}\ (P) = max\ \{\text{concavity}\ (x)\},\quad \forall\, x \in P \qquad (۳)$$

تعریف (۶) : چندضلعی p، τ-محدب است اگر تمام رئوس p تقعری کمتر یا مساوی τ داشته باشند [4,1]. به عبارت دیگر

$$\text{if}\ \forall\, v \in V\ \ \text{concavity}(v) \le \tau \ \rightarrow\ \text{concavity}\ (p) \le \tau \qquad (۴)$$

تعریف (۷) : به یک راس واقع در یک پاکت که بیشترین فاصله را از پل مربوط به آن پاکت (β) دارد، راس پاکت کمینه[13] گفته می‌شود.

$$\text{Pm}\ (p) = \{v_m \mid v_m \in p,\ \text{dist}\ (v_m, \beta) = \max_{v \in p} \text{dist}\ (v, \beta)\} \qquad (۵)$$

تعریف (۸) : تقعر نسبی RC برای برش c، نسبت تقعر مدل M قبل از اعمال برش c به تقعر آن بعد از اعمال برش c می‌باشد. به عبارت دیگر:

$$\text{RC (c)} = \frac{\text{Mb}}{\text{Ma}} \qquad (۶)$$

که در آن Mb، معرف concavity (M) و Ma، معرف بیشترین تقعر در بین اجزا می‌باشد.

شکل (۲) : مثالی از چند نمونه از پل‌ها و پاکت‌ها در یک چندضلعی

اندازه گیری تقعر روش‌های مختلفی دارد که در [4] آمده است. در FACD و در این مقاله از روش خط مستقیم استفاده شده است.

۳- تحلیل پیچیدگی زمانی FACD

در این بخش ما الگوریتم FACD را به چهار بخش تقسیم کرده و پیچیدگی زمانی آن‌ها را گام به گام تحلیل کرده‌ایم.

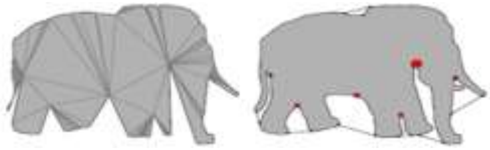
```
Algorithm FACD (M, r)
Input: A model M and tolerance r
Output: Components {Mi} of decomposed M
1: Find the potential cuts {Ck} in M
2: Build a cut_graph G from {Ck}
3: Use G and τ to select a set of cuts {Cr}
4: Apply {Cr} to decompose M into components {Mi}
5:    for each Mi in {Mi} do
6:       if concavity (Mi) ≥ τ then
7:          FACD (Mi, r)
8:       end if
9:    end for
```

الگوریتم (۱) : FACD [3]

۳-۱- تحلیل پیچیدگی زمانی گام اول الگوریتم FACD

این گام از دو قسمت ساده سازی چندضلعی و مثلث‌بندی دلونی تشکیل شده است. ابتدا باید رئوس پاکت کمینه (رئوسی که بیشترین تقعر را در هر پاکت دارند) تشخیص داده شوند. برای تشخیص این رئوس باید یک دور چندضلعی را پیمایش کنیم. رئوس پشت سر همی که روی چندضلعی قرار دارند، اما روی پوسته محدب قرار ندارند، نشان دهنده یک پاکت هستند. حال رئوس واقع در هر پاکت را پیمایش کرده و تقعر هر راس آن را محاسبه می‌کنیم تا راس‌های پاکت کمینه به دست آیند. از آن‌جا که در این گام یک‌بار پوسته محدب با هزینه $O(n \log n)$ محاسبه می‌شود [13] و یک پیمایش چندضلعی با هزینه $O(n)$ انجام می‌گیرد، هزینه زمانی این فرآیند حاصل جمع پیمایش چندضلعی و

تشکیل پوسته محدب است که از مرتبه $O(n \log n)$ می‌باشد. حال تمام رئوس واقع در هر پاکت حذف گردیده و از هر پاکت فقط رئوس ابتدایی و انتهایی پاکت و همچنین راس پاکت کمینه قرار می‌گیرد (که به این عمل ساده سازی چندضلعی می‌گویند). پس از این عمل چندضلعی بدست آمده را مثلث‌بندی دلونی کرده که هزینه این کار $O(n \log n)$ می‌باشد [2]. پس در مجموع هزینه زمانی این گام از مرتبه $O(n \log n)$ است. یال‌های بدست آمده از مثلث‌بندی دلونی را برش‌های بالقوه می‌گویند (شکل (۳)). که در گام سوم FACD با استفاده از برنامه نویسی پویا از بین این برش‌های بالقوه بهترین برش‌ها را برمی‌گزینند.



(الف) (ب)

شکل (۳) : (الف) رئوس پاکت کمینه (ب) برش های بالقوه (شکل، ساده‌سازی و سپس مثلث‌بندی دلونی شده است) [3]

۲-۳- تحلیل پیچیدگی زمانی گام دوم الگوریتم FACD

این گام شامل تشکیل گراف برش (گرافی که رئوس آن مثلث‌های دلونی و یال‌های آن با توجه به همسایگی این مثلث‌ها بدست می‌آید. بنابراین درجه هر راس در این گراف حداکثر سه می‌باشد زیرا هر مثلث حداکثر سه همسایه دارد) و یافتن یک ترتیب از رئوس گراف (یک ترتیب از مثلث‌های همسایه) می‌باشد. پس از مثلث‌بندی تعداد مثلث‌های تولید شده حداکثر n-2 می‌باشد. بنابراین تشکیل یک گراف با رئوسی از مثلث ها (حداکثر n-2 راس) که هر مثلث حداکثر سه همسایه داشته باشد، حداکثر هزینه زمانی معادل $3\times(n\text{-}2)$ خواهد داشت که از مرتبه زمانی $O(n)$ خواهد بود. برای یافتن یک ترتیب از رئوس گراف از پیمایش اول عمق استفاده شده است که این پیمایش به اندازه $O(n)$ زمان می‌برد. پس هزینه زمانی این گام از مرتبه $O(n)$ می‌باشد. تشکیل گراف برش در شکل (۵) نشان داده شده است.

۳-۳- تحلیل پیچیدگی زمانی گام سوم الگوریتم FACD

این گام شامل برنامه نویسی پویا برای انتخاب بهترین برش‌ها از بین برش‌های بالقوه‌ای که در گام اول و ترتیبی از مثلث‌ها که در گام دوم بدست آمده است، می‌باشد. به عبارت دیگر باید برش‌های مناسب از رابطه (۷) با استفاده از برنامه نویسی پویا پیدا شوند [3]. محاسبه S (i, j) ها به صورت قطری می‌باشد [3].

$$S(i, j) = \max \{S(i, k) + S(k + 1, j) + RC_{(k)}\}, \quad \forall\ i \le k < j\ ,\ S(i, i) = 0\ \ \forall\ 1 \le i < n \qquad (۷)$$

قضیه (۱) : در گام سوم الگوریتم FACD برای محاسبه هر $S(i, j)$ به ازای هر برش سه پوسته محدب محاسبه می‌شود.

اثبات: از آن‌جا که به ازای هر برش k روی یک چند ضلعی ساده دو چند ضلعی به وجود می‌آید (یکی چندضلعی سمت چپ برش k، دیگری چندضلعی سمت راست برش k) و یک چندضلعی هم کل چندضلعی بدون حضور برش k می‌باشد. پس در کل به ازای هر برش k، سه پوسته محدب در الگوریتم FACD محاسبه می‌شود.

قضیه (۲) : تعداد پوسته‌های محدبی که در گام سوم الگوریتم FACD محاسبه می‌شود، از مرتبه $O(n^3)$ می‌باشد.

اثبات: طبق رابطه (۷) برای محاسبه هر $S(i, j)$، $j\text{-}i$ تا برش k وجود دارد و طبق قضیه (۱) به ازای هر برش k، سه پوسته محدب محاسبه می‌شود. بنابراین در محاسبه عناصر قطر اول یعنی $S(i, i+1)$ ها فقط یک برش و برای محاسبه هر برش سه پوسته محدب محاسبه می‌شود و برای محاسبه عناصر قطر دوم یعنی $S(i, i+2)$ ها برای هر عنصر شش پوسته محدب محاسبه می‌شود و به همین ترتیب برای محاسبه قطر n-1 ام برای تنها عنصر آن یعنی $S(1, n)$، $3\times(n\text{-}1)$ پوسته محدب محاسبه می‌شود. بنابراین تعداد کل پوسته‌های محدب تولید شده در گام سوم الگوریتم FACD به صورت زیر محاسبه می‌شود (علامت $|x|$ به معنای تعداد مجموعه x می‌باشد):

$$\sum_{i=1}^{n-1} 3 \times (|i\ th\ diagonal's\ elements|) \times (|cuts|) = \sum_{i=1}^{n-1} 3 \times i \times (n-i)\ \in O(n^3) \qquad (۸)$$

قضیه (۳) : پیچیدگی زمانی گام سوم الگوریتم FACD از مرتبه $O(n^4 \log n)$ می‌باشد.

اثبات: عناصر $S(i, i+1)$ ها، یعنی هر یک از عناصر روی قطر اول در خلال برنامه نویسی پویا تنها یک برش را برسی می‌کنند، یا به عبارت دیگر بین یک مقدار ماکزیمم‌گیری می‌شود. برای محاسبه هر یک از عناصر قطر دوم یعنی $S(i, i+2)$ ها بین دو مقدار و به همین ترتیب برای عناصر قطر n ام بین n مقدار ماکزیمم‌گیری می‌شود. بنابراین هزینه هر قطر از حاصلضرب هزینه محاسبه پوسته محدب یعنی $O(n \log n)$ در تعداد عناصر موجود در آن قطر (قطر i ام، $(n\text{-}i)$ عضو دارد) ضربدر هزینه ماکزیمم‌گیری (اگر تعداد اجزایی که می‌خواهیم بین آن‌ها ماکزیمم‌گیری کنیم i باشد هزینه ماکزیمم‌گیری از مرتبه زمانی $O(i)$ می‌باشد [13]) ضربدر عدد سه (طبق قضیه (۱) به ازای هر برش سه پوسته محدب محاسبه می‌شود) به دست می‌آید. پس هزینه کل گام سوم الگوریتم FACD به صورت زیر محاسبه می‌شود:

$$\sum_{i=1}^{n-1}(the\ cost\ of\ a\ i\_th\ diameter)$$
$$\sum_{i=1}^{n-1} 3 \times O(n \log n) \times i \times (n-i)\ \in O(n^4 \log n) \qquad (۹)$$

۴- الگوریتم FACD بهبود داده شده (IFACD)

گام سوم الگوریتم FACD پر هزینه ترین بخش این الگوریتم است و با پیچیدگی زمانی $O(n^4 \log n)$ خود، زمان کل الگوریتم را تحت تاثیر قرار می‌دهد. در این گام تعداد پوسته‌های محدبی که محاسبه می‌شود از مرتبه $O(n^3)$ می‌باشد (قضیه (۲)) در حالی که تعداد کل پوسته‌های محدب ممکن، از مرتبه $O(n^2)$ می‌باشند. این هزینه زمانی نسبتا بالا بیشتر به خاطر محاسبات زیاد و تکراری پوسته‌های محدب می‌باشد. می‌توان برای کاهش هزینه این گام، قبل از ورود به گام سوم در یک مرحله پیش‌پردازش تمام پوسته‌های محدب ممکن را محاسبه و در یک آرایه ذخیره کرد و در خلال برنامه نویسی پویا و در مواقع نیاز با هزینه زمانی $O(1)$ از آرایه استفاده کرد. بنابراین قبل از ورود به گام سوم تمام پوسته‌های محدب غیر تکراری را محاسبه کرده و در یک آرایه ذخیره می‌کنیم.

```
Algorithm IFACD (M, r)
Input: A model M and tolerance r
Output: Components {Mi} of decomposed M
Find the potential cuts {Ck} in M
Build a cut_graph G from {Ck}
Compute all possible convex hulls and store in array
Use G and τ to select a set of cuts {Cr}
Apply {Cr} to decompose M into components {Mi}
  for each Mi in {Mi} do
     if concavity (Mi) ≥ τ then
          IFACD (Mi, r)
     end if
end for
```

الگوریتم (۲) : IFACD

قضیه (۴) : در گام سوم الگوریتم IFACD هر $S(i, j)$ با هزینه $O(n)$ محاسبه می‌شود.

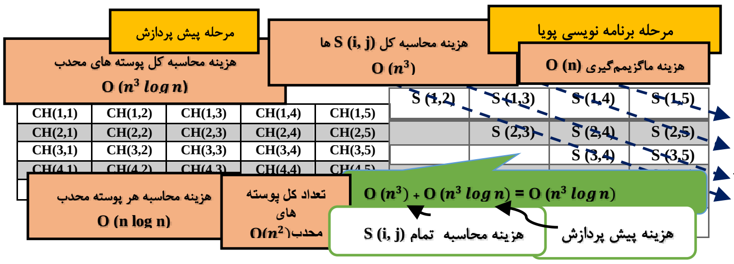
اثبات: از آن‌جا که در مرحله پیش‌پردازش گام سوم IFACD، تمام پوسته‌های محدب محاسبه شده‌اند، پس هزینه دسترسی به آرایه را داریم که از مرتبه $O(1)$ می‌باشد و هر $S(i, j)$ با هزینه زمانی $O(n)$ محاسبه می‌شود که آن هم هزینه ماکزیمم‌گیری می‌باشد (شکل (۴)).

قضیه (۵) : تعداد کل پوسته‌های محدبی که در الگوریتم IFACD استفاده شده است، حداکثر از مرتبه $O(n^2)$ می‌باشد.

اثبات: فرض کنید حداکثر *n-2* مثلثی را که از گام اول بدست آمده است، به گراف برش با حداکثر *n-2* راس تبدیل کرده و با یک پیمایش ترتیب O_1 از رئوس این گراف بدست آمده باشد. رئوس بدست آمده برحسب این ترتیب را، به صورت *{1, 2, 3, …, m}* نام‌گذاری می‌کنیم (*m* حداکثر می‌تواند $3\times 2 \times n$ باشد زیرا در گراف برش هر راس حداکثر سه همسایه دارد که برای پیمایش اول عمق این سه همسایه هر یک را پیموده و به راس اولیه باز می‌گردد و سپس به پیمایش همسایه بعدی می‌رود که این کار برای هر راس با سه همسایه حداکثر شش بار (3×2) انجام می گیرد). هر زیر مسئله از برنامه نویسی پویا یک دنباله پشت سر هم از ترتیب O_1 می‌باشد [3]. پس تمام زیر مسئله‌هایی که در برنامه نویسی پویا تولید می‌شوند، یک دنباله پشت سر هم از مجموعه رئوس پیمایش O_1 می‌باشند. از آن‌جا که *m* حداکثر $6\times n$ می‌باشد، یافتن تمام دنباله های پشت‌سرهم از یک مجموعه $6\times n$ عضوی، از مرتبه $O(n^2)$ می‌باشد. بنابراین تمام پوسته‌های محدبی که در IFACD محاسبه می‌شوند، از مرتبه $O(n^2)$ می‌باشد.

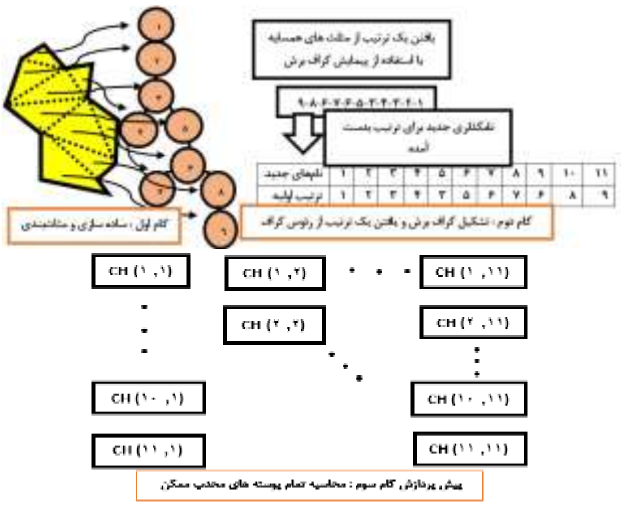
قضیه (۶) : پیچیدگی زمانی گام سوم الگوریتم IFACD از مرتبه $O(n^3 \log n)$ می‌باشد.

اثبات: از آن‌جا که هزینه محاسباتی هر پوسته محدب از مرتبه زمانی $O(n \log n)$ می‌باشد [13]، پس هزینه زمانی این پیش پردازش از مرتبه $O(n^3 \log n)$ می باشد (زیرا طبق قضیه (۵) تعداد پوسته‌های محدبی که باید محاسبه شوند از مرتبه $O(n^2)$ است). از آن‌جا که تعداد $S(i, j)$ ها، $\frac{n^2-n}{2}$ است پس هزینه محاسبه کل $S(i, j)$ ها برابر $O(n^3)$ می‌باشد (زیرا طبق قضیه (۴) هر $S(i, j)$ با هزینه $O(n)$ محاسبه می‌شود). با توجه به هزینه پیش پردازش انجام شده و هزینه محاسبه $S(i, j)$ ها، هزینه کلی گام سوم IFACD از مرتبه $O(n^3 \log n)$ می‌باشد. این بهبود در شکل (۴) نشان داده شده است.



شکل (۴) : بهبود گام سوم الگوریتم FACD به شکل یک مثال

مثال (۱) : جزئیات الگوریتم IFACD در شکل (۵) آمده است:



شکل (۵) : روند الگوریتم IFACD

۵- نتیجه

پرهزینه‌ترین بخش الگوریتم FACD که زمان محاسباتی تمام الگوریتم را تحت تاثیر خود قرار می‌دهد، گام سوم آن است. در این مقاله با بهبودی که به صورت یک پیش‌پردازش برای گام سوم الگوریتم FACD محاسبه شد، مرتبه زمانی گام سوم این الگوریتم از مرتبه $O(n^4\ log\ n)$ به مرتبه $O(n^3\ log\ n)$ کاهش یافت. از آنجا که ممکن است در گام سوم این الگوریتم بعضی از پوسته‌های محدب محاسبه شده در مرحله پیش پردازش نیز بدون استفاده باقی بمانند، می‌توان روشی را ارائه کرد که با بهینه سازی در مرحله پیش پردازش، روی محاسبه $O(n^2)$ پوسته محدب، فقط پوسته‌های محدب مورد نیاز محاسبه شود و از محاسبه پوسته‌های محدب اضافی نیز رها شد. هر چند که این کار هزینه کلی الگوریتم IFACD را کاهش نمی‌دهد اما به نوبه خود می‌تواند در تسریع الگوریتم تاثیر گذار باشد.

مراجع

[1] J. Lien and N. Amato, "Approximate convex decomposition for Polygons," *Computational Geometry Theory & Applications,* vol. 35, pp. 100-123, 2006.

[2] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams," *ACM Transactions on Graphics,* p. 75–123, 1985.

[3] M. Ghosh, M. Amato, Y. Lu and J. Lien, "Fast Approximate Convex Decomposition," *Computer-Aided Design,* vol. 45, pp. 494-504, 2013.

[4] J. Lien, "Approximate convex decomposition and its application," *Ph.D. dissertation. Texas A & M University, College Station,* 2006.

[5] L. Fernandez and B. Canvas, "Algorithms for the decomposition of a polygon into convex polygons," *European Journal of Operational Research,* vol. 121, pp. 330-342, 2000.

[6] K. Siddiqi and B. Kimia, "Parts of visual form," *Computational aspects,* vol. 1, 1995.

[7] M. Cohen-Steiner, P. Alliez and M. Desbrun, "Variational shape approximation," *In International Conference on Computer Graphics and Interactive Techniques,* p. 905–914, 2004.

[8] X. Li, T. Woon, T. Tan and Z. Huang, "Decomposing polygon meshes for interactive applications," *In Proceedings of the 2001 symposium on Interactive 3D graphics,* pp. 35-42, 2001.

[9] H. Liu, L. Latecki and W. Lio, "Convex shape decomposition.," *In Proc. of IEEE Computer Vision and Pattern Recognition,* pp. 104-124, 2010.

[10] B. Chazelle, "A theorem on polygon cutting with applications," *IEEE Sympos,* pp. 339-349, 1982.

[11] D. Greene, "The decomposition of polygons into convex parts," *In Computational Geometry,* vol. 1, pp. 235-259, 1983.

[12] J. Keil, "Decomposing a polygon into simpler components," *SIAM J. Comput,* vol. 14, pp. 799-817, 1985.

[13] G. Selim, The Design and Analysis of Parallel Algorithm, vol. 401, New jersey: Prentice Hall, Englewood Cliffs, 1989.

[14] J. Keil and J. Snoeyink, "On the time bound for convex decomposition of simple polygons," in *In Proceedings of the 10th Canadian Conference on Computational Geometry,* Quebec, Canada: School of Computer Science, McGill University, 1998.

[15] B. Chazelle and D. Dobkin, "Optimal convex decompositions," *Computational Geometry,* pp. 63-133, 1985.

زیر نویس‌ها

---

[1] Fast Approximate Convex Decomposition
[2] Improved Fact Approximate Convex Decomposition
[3] Shape Analysis
[4] Understanding
[5] Shape Simplification
[6] Skeleton Extraction
[7] Meaningful Parts
[8] Visuality
[9] Dynamic Programing
[10] Exactly Convex Decomposition
[11] Approximate Convex Decomposition
[12] Chain
[13] Pocket Minima