دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه دکتری
گرایش نرم‌افزار

عنوان
الگوریتم‌های ساده‌سازی هندسی

نگارش
شروین دانش‌پژوه

استاد راهنما
دکتر محمد قدسی

بهمن ۱۳۹۲

دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

رساله دکتری

الگوریتم‌های ساده‌سازی هندسی

نگارش: شروین دانش‌پژوه

استاد راهنما: دکتر محمد قدسی امضاء:

استاد ممتحن داخلی: دکتر امضاء:

استاد ممتحن خارجی: دکتر امضاء:

چکیده

در این پایان نامه به مسئله ساده‌سازی مسیر می‌پردازیم و الگوریتم‌های مختلفی را برای آن ارایه می‌کنیم. ابتدا مسئله را تحت معیار مساحت بررسی می‌نماییم و یک الگوریتم تقریبی برای معیار جمع- مساحت و یک الگوریتم بهینه برای معیار تفاضل- مساحت ارایه می‌دهیم. الگوریتم‌های فوق بر روی مسیرهای عمومی قابل اعمال هستند. در ادامه مسئله ساده‌سازی هم‌فضای مسیر را بررسی می‌کنیم. ما چارچوبی را برای حل این مسئله ارایه می‌دهیم. با استفاده از این چارچوب می‌توان مسئله را تحت هر معیار دلخواه حل نمود. ابتدا مسئله ساده‌سازی قویا- هم‌فضا را برای مسیرهای x- یکنوا بررسی می‌نماییم و یک الگوریتم بهینه برای آن ارایه می‌دهیم. سپس مسئله را برای مسیرهای عمومی بررسی می‌کنیم. برای حل این مسئله ما یک داده‌ساختار درختی هندسی ارایه می‌دهیم و با استفاده از این داده‌ساختار مسئله را به صورت بهینه حل می‌کنیم. در ادامه مسئله ساده‌سازی هم‌فضا را برای مسیرهای عمومی بررسی نموده و یک الگوریتم بهینه برای آن ارایه می‌دهیم. تا آن جایی که ما می‌دانیم دو الگوریتم اخیر، اولین الگوریتم‌هایی هستند که این دو مسئله را به صورت بهینه حل می‌کنند. همچنین، برای کاربردهایی که زمان اجرا در آن‌ها اهمیت دارد یک الگوریتم مکاشفه‌ای ارایه می‌دهیم. سپس مسئله ساده‌سازی را تحت معیار قابلیت دید بررسی می‌کنیم. ابتدا این مسئله را تحت معیار اندازه ضلع‌های قابلیت دید و بیشینه خطای مسیر بررسی می‌نماییم و یک الگوریتم چند جمله‌ای برای آن ارایه می‌دهیم. در نهایت مسئله ساده‌سازی مسیر را، تحت معیار مساحت قابلیت دید و مجموع خطای مسیر بررسی می‌کنیم. ما نشان می‌دهیم این مسئله انپی- سخت است و یک الگوریتم تقریبی برای آن ارایه می‌دهیم.

کلمات کلیدی: *هندسه محاسباتی، ساده‌سازی مسیر، هم‌فضا ، مساحت، قابلیت دید*

تقدیم به

مادرم و پدرم

و همسرم

که وجودشان مایه سعادت

و

عشقشان مایه استواری است.

قدردانی

در این جا لازم می‌دانم از استاد گرانقدر جناب آقای دکتر محمد قدسی که از دوره کارشناسی ارشد تاکنون و در همه مراحل دوره دکتری با مساعدت، حمایت و راهنمایی‌های بی‌دریغ و بزنگاه خود اینجانب را یاری کردند، تشکر و قدردانی نمایم. همچنین لازم می‌دانم از اساتید و عزیزانی که در این تز همکاری داشتند تشکر نمایم؛ بخصوص از آقای دکتر محمد آبام برای راهنمایی‌ها و کمک‌های ارزنده‌شان در طی دوره فرصت مطالعاتی و پس از آن در ویرایش‌های متعدد مقاله ژورنال، آقای دکتر زارعی، آقای دکتر لسه دلوران، آقای مهندس شایان احسانی، آقای مهندس محسن محمودی ازناوه، آقای مهندس میلاد غلامی، مهندس سینا کشتکار جعفری و آقای مهندس مجتبی نوری بایگی. از آقای دکتر لارس آرگه برای پذیرش اینجانب برای دوره فرصت مطالعاتی تشکر می‌نمایم. رفتن به فرصت مطالعاتی میسر نبود مگر با ضمانت آقای دکتر اکبری نوری و آقای مهندس ابوذر رحمتی. مجددا از ایشان تشکر می‌نمایم. در این جا باید از آقای دکتر حمید سربازی آزاد و آقای دکتر سعید گرگین برای فراهم آوردن فرصت آشنایی و بهره‌مندی از امکانات مرکز پردازش سریع پژوهشگاه دانش‌های بنیادی تشکر نمایم. فرصت را غنیمت شمرده صمیمانه از دوستانم که بعضا کار تحقیقاتی مشترک هم داشته‌ایم (به ترتیب حروف الفبا) آقای مهندس محمد توکلی قینانی، آقای دکتر حسین جوهری، آقای مهندس کامیار خدامرادی، آقای مهندس زاهد رحمتی، خانم مهندس شراره علیپور، آقای مهندس محمد حسین غفاری انجدانی، آقای مهندس مجتبی محمدی نصیری، آقای دکتر مرتضی منعمی‌زاده، آقای دکتر مصطفی نوری بایگی و آقای مهندس حامد یعقوبی شهیر برای کمک‌ها و همراهی‌هایشان تشکر می‌نمایم.

این پایان نامه به وسیله نرم‌افزار زی‌پرشین و لاتک تهیه شده است. همین جا از تهیه‌کنندگان و توسعه‌دهندگان آن قدردانی می‌نمایم.

فهرست مطالب

فهرست شکل‌ها

فهرست جدول‌ها

فصل ۱

مقدمه

ساده‌سازی هندسی یکی از مسایلی است که در زمینه‌های مختلف مرتبط با کامپیوتر، مانند سامانه‌های اطلاعات جغرافیایی (GIS)[۱]، گرافیک کامپیوتری، بینایی ماشین، پردازش تصاویر، تشخیص الگوها، آمار و فشرده‌سازی داده‌ها کاربردهای بسیاری دارد [۱، ۲، ۳، ۴، ۵، ۶، ۷، ۸، ۹]. در اغلب کاربردها، شیء هندسی عموما به صورت مسیر نمایش می‌یابد. به عنوان نمونه، شیء هندسی در سیستم‌های اطلاعات جغرافیایی، می‌تواند نقشه یک کشور باشد. روشن است که نقشه یک کشور در مقیاس واقعی، برای انسان قابل خواندن نیست. بنابراین طبیعی است که باید، ضمن حفظ شکل کلی نقشه، آن را کوچک نموده تا برای انسان قابل استفاده باشد. به این عمل اصطلاحا ساده‌سازی گفته می‌شود. معمولا، در ساده‌سازی از جزئیات چشم پوشی و الگویی کلی و با کیفیت از شکل اصلی ارایه می‌شود. ساده‌سازی علاوه بر کاربرد در ساده‌سازی نقشه، در کاربردهای دیگر که اطلاعات به صورت مسیرهای چندضلعی نمایش می‌یابند، نیز کاربرد دارد. به عنوان نمونه می‌توان به ساده‌سازی مسیر حرکت کامیون‌های باری، مسیر پرواز پرندگان مهاجر و غیره اشاره نمود. در این پایان نامه ما الگوریتم‌های جدیدی را برای مسئله ساده‌سازی مسیر ارایه می‌دهیم.

۱.۱ انگیزه‌ی پژوهش

مسئله ساده‌سازی مسیر در ادبیات این موضوع به نام‌های ساده‌سازی زنجیره چندضلعی، مسیر چندضلعی، منحنی و خط شناخته می‌شود [۱۰، ۱۱، ۱۲، ۱۳، ۱۴، ۱۵، ۱۶، ۱۷]. مسیرها معمولا با تعداد زیادی نقطه مشخص می‌شوند. به دلیل حجم زیاد داده‌های ورودی، پردازش آن‌ها کاری چالش برانگیز است. اغلب برای پردازش و نمایش بهتر چنین مسیرهایی، آن‌ها را با روش‌های ساده‌سازی مسیرها ساده می‌کنند. در این روش‌ها مسیر اصلی

[۱] Geographic Information System

(ورودی) به مسیر جدیدی که دارای تعداد نقطه‌های کمتری نسبت به مسیر اصلی است و در عین حال تقریب مناسبی از مسیر اصلی به دست می‌دهد، تبدیل می‌شود. تعریف دقیق مسئله در بخش ۱.۲ آمده است.

یکی از مهمترین کاربردهای این مسئله در نقشه‌کشی[۲] است. بیشتر خصوصیات نقشه‌ها یا مستقیما توسط مسیرهای مختلف نمایش می‌یابند (مانند خطوط راه‌ها، رودخانه‌ها) یا آن که توسط یک چندضلعی که با خطوط محدود شده است (مانند ناحیه‌های نظامی، مرز کشورها، خطوط ساحلی و مناطق حفاظت شده) نشان داده می‌شوند. در این مسئله هدف ارایه مسیرهای ساده‌تر است، به‌گونه‌ای که شکل کلی مسیرها حفظ شود. در برخی کاربردها علاوه بر اطلاعات مرزها، اطلاعات اضافی دیگری مانند محل شهرها، ایستگاه‌های قطار، معادن و ... نیز داده می‌شود. در این حالت، ساده‌سازی باید به گونه‌ای انجام شود که پس از انجام ساده‌سازی محل یا سمت شیءهای مشخص شده، تغییر نکنند. به این گونه از ساده‌سازی، ساده‌سازی هم‌فضا [۳] می‌گویند. در برخی موارد لازم است که هر پاره‌خط از مسیر ساده شده نسبت به زیرمسیر متناظر در مسیر اصلی نیز هم‌فضا باشد. این مسئله را ساده‌سازی قویا-هم‌فضا می‌نامیم (برای تعریف دقیق آن به بخش ۴.۲ مراجعه شود). به عنوان نمونه‌ای از کاربرد این مسئله در رباتیک، به مثالی که در ادامه می‌آید توجه کنید. رباتی را در نظر بگیرید که در ساختمانی که دارای ستون‌های عمودی است، قرار دارد. ربات روی مسیر از پیش تعریف شده‌ای حرکت نموده و با استفاده از ابزارهایی که در یک واگن است، به تعمیر برخی اشکالات در ساختمان می‌پردازد. جابجایی واگن پرهزینه است و ما مایل هستیم که این هزینه را کمینه کنیم. همچنین اگر فاصله ربات از واگن بیش از اندازه مشخص ϵ باشد، ربات نمی‌تواند به آن دسترسی داشته باشد. ما به دنبال مسیر کوتاه‌تری برای جابجایی واگن هستیم به طوری که فاصله آن از ربات کمتر از ϵ بوده و هر ضلع از مسیر ساده شده واگن، نسبت به زیرمسیر متناظر در مسیر اصلی، که ربات روی آن حرکت می‌کند، قویا-هم‌فضا باشد. توجه داشته باشید که اگر ستونی بین این ضلع و زیرمسیر متناظر باشد، ربات نمی‌تواند بازوهایش را برای برداشتن ابزار از واگنِ ابزار حرکت دهد. در واقع این کاربرد، مسئله ساده‌سازی قویا-هم‌فضای مسیر تحت معیار خطای فرشه[۴] است.

به عنوان یک کاربرد دیگر این مسئله در رباتیک، یک ربات نگهبان را در درون یک ساختمان در نظر بگیرید. این ربات وظیفه دارد بر روی مسیر از پیش تعیین شده‌ای حرکت کند و در حین حرکت، شیءهای ناشناس را شناسایی و گزارش کند. در این مسئله ما به دنبال مسیر کوتاه‌تر با قابلیت دید نزدیک به قابلیت دید مسیر اصلی هستیم (برای تعریف دقیق آن بخش ۹.۲ را ببینید).

از دیگر کاربردهای این روش می‌توان به ساده‌سازی مرزها یا شکل‌های هندسی اشاره نمود. در این حالت مساحت جابجا شده در ساده‌سازی اهمیت پیدا می‌کند. علاوه بر این در رباتیک، از این روش برای ساده‌سازی

[۲] Cartographic

[۳] Homotopic

[۴] Fréchet

و حذف نویز در داده‌های به دست آمده از پویشگرهای لیزری نیز استفاده می‌شود. همچنین باید به کاربرد این روش در آمار و تحلیل برازش اشاره نمود. برازش خطی، یک مجموعه نقطه را با یک خط تقریب می‌زند و برازش غیرخطی آن را با یک تابع غیرخطی تقریب می‌زند. هدف تکنیک برازش منحنی این است که تقریبی برای منحنی داده شده در صفحه به دست آورد.

۲.۱ نتیجه‌های به دست آمده

در ادامه نتیجه‌های به دست آمده آورده شده‌اند (برای آشنایی با تعاریف و اصطلاح‌هایی که در ادامه استفاده می‌شود به فصل ۲ مراجعه نمایید):

- الگوریتم‌های بهینه و تقریبی برای ساده‌سازی مسیر مبتنی بر معیار جمع-مساحت
- الگوریتم بهینه برای ساده‌سازی مسیر مبتنی بر معیار تفاضل مساحت
- چارچوب بهینه برای مسئله ساده‌سازی قویا-هم‌فضای مسیرهای x-یکنوا
- چارچوب بهینه برای مسئله ساده‌سازی قویا-هم‌فضای مسیرهای عمومی
- چارچوب بهینه برای مسئله ساده‌سازی هم‌فضای مسیرهای عمومی
- الگوریتم مکاشفه‌ای برای مسئله ساده‌سازی هم‌فضای مسیرهای عمومی
- الگوریتم ساده‌سازی مسیر تحت معیار تعداد ضلع‌های قابلیت دید و بیشینه خطای مسیر
- اثبات ان‌پی-سخت بودن مسئله ساده‌سازی تحت معیار مساحت قابلیت دید و مجموع خطای مسیر
- الگوریتم تقریبی برای مسئله ساده‌سازی تحت معیار مجموع مساحت قابلیت دید و مجموع خطای مسیر

در ادامه به اختصار به توضیح نتیجه‌های فوق می‌پردازیم.

۱.۲.۱ ساده‌سازی تحت معیارهای مساحت

برای ساده‌سازی تحت معیارهای مساحت، ما ابتدا یک تعریف یکپارچه برای معیار جمع-مساحت و تفاضل-مساحت ارایه می‌دهیم به طوری که بر روی مسیرهای عمومی قابل اعمال باشد. سپس با استفاده از این تعریف به ارایه الگوریتم تحت معیارهای جمع-مساحت و تفاضل-مساحت می‌پردازیم.

الگوریتم‌های بهینه و تقریبی برای ساده‌سازی مسیر مبتنی بر معیار جمع-مساحت

در این مسئله، مسیر $\mathcal{P}$ شامل n نقطه را به عنوان ورودی دریافت می‌کنیم و هدف این است که مسیر تقریبی $\mathcal{Q}$ را تحت معیار خطای جمع-مساحت، به دست آوریم. مسئله ساده‌سازی مسیر تحت معیار جمع-مساحت اولین بار توسط مک‌مستر [۱۸، ۲] مطالعه شد. سپس ویزوالینگام و وایات [۱۹] یک الگوریتم ساده سازی خطی مبتنی بر *ناحیه مؤثر* ارایه دادند. اولین تحلیل الگوریتمی برای مسئله ساده‌سازی مسیر تحت معیار جمع-مساحت، را بوز و همکارانش [۱۶] ارایه کردند. آن‌ها الگوریتمی برای مسئله ϵ-کمینه و تحت معیار جمع-مساحت و مجموعه خطای مسیر با پیچیدگی زمانی $O(n^{۲+\epsilon})$ ارایه دادند. الگوریتم‌های ارایه شده آن‌ها تنها بر روی مسیرهای x-یکنوا کار می‌کند. ما ابتدا با استفاده از تعریف یکنواختی که ارایه دادیم و همچنین استفاده از الگوریتم عمومی ایمایی و ایری [۱۰]، یک الگوریتم ابتدایی با پیچیدگی زمانی $O(n^{۳})$ ارایه می‌کنیم. سپس یک الگوریتم تقریبی، که دارای پیچیدگی زمانی کمتری نسبت به الگوریتم ارایه شده قبلی است، ارایه می‌دهیم. ایده الگوریتم تقریبی ارایه شده این است که از نتیجه‌ی محاسبه‌ی خطای هر میانبر برای محاسبه کارای خطای میانبر بعدی استفاده شود. این کار با استفاده از محاسبه و نگهداری خطای زیرمسیر فعلی برای مجموعه‌ای از خطوط کانونی که از گره آغازین منشعب می‌شوند، انجام می‌شود. برای هر نقطه بعدی، ما دو خط کانونی که این نقطه ما بین آن‌ها است را می‌یابیم و خطای میانبر جدید را با استفاده از خطای خطوط کانونی محاسبه می‌کنیم. ما نشان دادیم که این الگوریتم در زمان $O(\frac{n^{۲}}{\epsilon})$ مسئله $\#$-کمینه را برای معیار جمع-مساحت و مجموع خطای مسیر و با خطای ساده‌سازی $O(\epsilon)$ حل می‌کند.

الگوریتم بهینه برای ساده‌سازی مسیر مبتنی بر معیار تفاضل-مساحت

در برخی کاربردها، مانند ساده‌سازی مرزها، مساحت جابجا شده در مسیر ساده شده نسبت به مسیر اصلی، اهمیت پیدا می‌کند. اولین نتیجه‌ها در زمینه معیار تفاضل-مساحت توسط بوز و همکارانش [۱۶] ارایه شد. آن‌ها نشان دادند که مسئله ساده‌سازی مسیر تحت معیار تفاضل-مساحت و برای مجموع خطای مسیر یک مسئله ان‌پی-سخت است. همچنین، آن‌ها برای مسئله ϵ-کمینه و برای معیار تفاضل-مساحت و خطای بیشینه مسیر الگوریتمی تقریبی با پیچیدگی زمانی $O(n^{۲+\epsilon})$ ارایه دادند. این الگوریتم نیز تنها بر روی مسیرهای x-یکنوا کار می‌کند.

ما الگوریتمی برای محاسبه خطای تفاضل-مساحت کلیه میانبرها ارایه می‌دهیم. پیچیدگی زمانی این الگوریتم برای مسیر $\mathcal{P}$ شامل n نقطه و $O(n^{۲})$ میانبر از مرتبه‌ی $O(n^{۲})$ است. در ادامه با استفاده از چارچوب ایمایی و ایری [۱۰] الگوریتمی برای ساده‌سازی مسیر با همین مرتبه زمانی ارایه می‌دهیم. علاوه بر بهبود زمان اجرا، دیگر نقطه قوت الگوریتم‌های ارایه شده این است که این الگوریتم بر روی مسیرهای عمومی قابل استفاده

است.

۲.۲.۱ الگوریتم‌های ساده‌سازی هم‌فضای مسیر

الگوریتم‌های بسیار زیادی برای ساده‌سازی مسیر تحت معیارهای مختلف ارایه شده است. اما تعداد معدودی از آن‌ها قادر به حفظ جهت شیءها (مانع‌ها)، و ارایه مسیر ساده‌شده به گونه ای که با مسیر اصلی هم‌فضا باشد، هستند. بهترین الگوریتم قبلی در این زمینه توسط دی‌برگ و همکارنش در سال ۱۹۹۵ ارایه شده است [۲۰، ۲۱]. آن‌ها الگوریتمی با پیچیدگی زمانی $O(n(n+m)\log n)$ ارایه نمودند که بر روی مسیرهای x-یکنوا کار می‌کند. همچنین، آن‌ها این الگوریتم را برای مسیرهای عمومی گسترش دادند. ولی این الگوریتم، بهینه بودن اندازه پاسخ را تضمین نمی‌کند. در این زمینه گویباس و همکارانش ثابت نمودند که برخی گونه‌های ساده‌سازی مسیر در نقشه‌ها دارای پیچیدگی ان‌پی-سخت است [۲۲]. استوسکی و میتچل نشان دادند که ساده‌سازی هم‌فضای k مسیر نیز ان‌پی-سخت است [۲۳]. ما برای گونه‌های مختلف مسئله ساده‌سازی هم‌فضای مسیر الگوریتم‌هایی را ارایه نمودیم. در واقع ما چارچوبی را برای این مسئله ارایه می‌دهیم که قادر به ساده‌سازی هم‌فضا، مستقل از معیار خطا است. به عبارت دیگر می‌توان این چارچوب را برای ساده‌سازی مسیر تحت هر معیار خطای دلخواهی مورد استفاده قرار داد. در ادامه به نتیجه‌های به دست آمده ما اشاره می‌کنیم.

چارچوب بهینه برای مسئله ساده‌سازی قویا- هم‌فضا برای مسیرهای x-یکنوا

مسیر $\mathcal{P}$ شامل n نقطه را مسیری x-یکنوا بر روی صفحه در نظر بگیرید. همچنین S را مجموعه‌ای شامل m نقطه به عنوان مانع در صفحه در نظر بگیرید. فرض کنید زمان مورد نیاز برای محاسبه G_ϵ تحت تابع خطای F در غیاب مانع‌ها، $T_F(n)$ باشد. نشان می‌دهیم که برای چنین مسیری، ساده‌سازی قویا-هم‌فضای بهینه را می‌توان در زمان $T_F(n)+O(m\log(m+n)+n\log n\log(n+m)+k)$، که در آن k تعداد میانبرهای هم‌فضا است، محاسبه نمود. روشن است که زمان $T_F(n)$ وابسته به معیار خطای F خواهد بود و الگوریتم وابستگی به معیار خطا ندارد. این الگوریتم بهترین الگوریتم قبلی [۲۰، ۲۱] را بیش از $\log n$ مرتبه بهبود می‌بخشد.

چارچوب بهینه برای مسئله ساده‌سازی قویا- هم‌فضا برای مسیرهای عمومی

مسیر عمومی $\mathcal{P}$ شامل n نقطه و مجموعه S شامل m مانع را در صفحه در نظر بگیرد. همچنین $T_F(n)$ را زمان لازم برای محاسبه‌ی G_ϵ تحت معیار خطا (تابع فاصله) F در حضور مانع‌ها نظر بگیرید. ما چارچوبی را ارایه می‌دهیم که ساده‌سازی قویا-هم‌فضای بهینه را در زمان $T_F(n)+O(n(m+n)\log(n+m))$ محاسبه می‌نماید. ایده این الگوریتم مبتنی بر تبدیل مسیر عمومی به مسیرهای x-یکنوا و سپس فشرده‌کردن این مسیرها

با توجه به مانع‌ها است. با استفاده از داده‌ساختاری که ارایه می‌دهیم نشان می‌دهیم که این فشرده‌سازی را می‌توان به گونه‌ای انجام داد که به جای ذخیره کردن کلیه زیرمسیرهای ممکن، آن‌ها را به صورت ضمنی نگهداری کنیم. در نهایت ما نشان می‌دهیم که کلیه $O(n^۲)$ میانبر ممکن را می‌توان به صورت کارا با زیرمسیرهای متناظر، از نظر هم‌فضا بودن، مقایسه نمود. این الگوریتم نه‌تنها بهترین الگوریتم قبلی [۲۰، ۲۱] را بیش از $\log n$ مرتبه بهبود می‌بخشد، بلکه، پاسخ بهینه را نیز ارایه می‌دهد.

چارچوب بهینه برای مسئله ساده‌سازی هم‌فضای مسیرهای عمومی

مسیر چندضلعی $\mathcal{P}$ با اندازه n را در صفحه‌ای حاوی m نقطه به عنوان مانع را در نظر بگیرید. همچنین $T_F(n)$ را زمان لازم برای محاسبه‌ی G_ϵ تحت معیار خطا (تابع فاصله) F در حضور مانع‌ها نظر بگیرید. ما چارچوبی را ارایه می‌دهیم که ساده‌سازی هم‌فضای بهینه را در زمان $O(n^۶m^۲) + T_F(n)$ و فضای $O(n^۴m^۲)$ به دست می‌آورد. در حل این مسئله ما از برنامه‌ریزی پویا بهره می‌بریم و یک الگوریتم پویای نسبتا پیچیده ارایه می‌دهیم. لازم به ذکر است که این الگوریتم، اولین الگوریتمی است که بر روی مسیرهای عمومی کار می‌کند و مسئله را به صورت بهینه حل می‌کند. به علاوه، این الگوریتم هم مانند دو الگوریتم بالا، مستقل از معیار خطا است.

الگوریتم مکاشفه‌ای برای مسئله ساده‌سازی هم‌فضای مسیرهای عمومی

برای بهبودِ زمان اجرای الگوریتمی که برای ساده‌سازی هم‌فضای مسیرهای عمومی ارایه دادیم، ما یک الگوریتم مکاشفه‌ای ارایه می‌کنیم. مسیر چندضلعی $\mathcal{P}$ با اندازه n را در صفحه‌ای حاوی m نقطه به عنوان مانع را در نظر بگیرید. الگوریتم مکاشفه‌ای ما قادر به شناسایی زیرمجموعه‌ای از میانبرهای هم‌فضای p_ip_j در زمان $O((n+m)\log(n+m+T_F(n))$ است، که در آن $T_F(n)$ زمان لازم برای محاسبه‌ی G_ϵ تحت معیار خطای F در حضور مانع‌ها هست. در واقع، این الگوریتم تضمین می‌کند که خروجی ساده شده هم‌فضا باشد. ولی، لزوما طول مسیر ساده‌شده کمینه نخواهد بود. برای بسیاری از کاربردهای عملی زمان اجرا بسیار اهمیت دارد. بنابراین، این روش با توجه زمان اجرای پایین آن (نسبت به الگوریتم بالا)، برای کاربردهایی که زمان در آن‌ها عامل مهم‌تری به حساب می‌آید، مناسب است.

۳.۲.۱ الگوریتم‌های ساده‌سازی تحت معیار قابلیت دید

مسیر $\mathcal{P}$ با اندازه n را در داخل چندضلعی ساده $\mathcal{R}$ با اندازه m در نظر بگیرید. می‌خواهیم مسیر را به گونه‌ای ساده کنیم که قابلیت دید مسیر ساده شده نزدیک به قابلیت دید مسیر اصلی باشد. ما دو گونه از این مسئله را تحت معیارهای اندازه قابلیت دید و مجموع مساحت قابلیت دید بررسی می‌کنیم. این مسئله در رباتیک و مسایل

مرتبط با قابلیت دید کاربرد دارد (برای مثالی در این زمینه، بخش ۹.۲ را ببینید).

الگوریتم ساده‌سازی تحت معیار تعداد ضلع‌های قابلیت دید

ما ابتدا مسئله را برای بیشینه خطای مسیر تحت معیار تعداد ضلع‌های قابلیت دید بررسی می‌کنیم. این مسئله را با یک راه حل ابتدایی می‌توان در زمان $O(n^۲m)$ حل نمود. ما نشان می‌دهیم که می‌توان، با انجام پیش‌پردازش بر روی $\mathcal{R}$ در زمان $O(m^۷)$ و با استفاده از حافظه $O(m^۶)$، این مسئله را برای یک پرسمان مسیر $\mathcal{P}$ با اندازه $O(n)$ در $O(n(n+m))$ زمان حل نمود. در حل این مسئله ما با سه زیر مسئله مواجه هستیم: (آ) یافتن چندضلعی قابلیت دید ضعیف برای کلیه زیرمسیرهای $\mathcal{P}(i,j)$، $۱ \leq i < j \leq n$، (ب) یافتن چندضلعی قابلیت دید ضعیف برای کلیه میانبرهای p_ip_j، $۱ \leq i < j \leq n$، و (ج) محاسبه خطای تعداد ضلع‌های قابلیت دید برای کلیه میانبرهای p_ip_j، $۱ \leq i < j \leq n$. با اثبات تعدادی لم و استفاده از برنامه‌ریزی پویا، ما نشان می‌دهیم که مسئله (آ) و (ج) را می‌توان بدون پیش پردازش در زمان $O(n(n+m))$ حل نمود. برای آن که اندازه قابلیت دید یک پاره‌خط را در زمان کمتر از $O(m)$ به دست آوریم، ما روشی ارایه می‌دهیم که با پیش پردازش $O(m^۷)$ و حافظه $O(m^۶)$، به چنین پرسمان‌هایی در زمان $O(۱)$ پاسخ دهد.

اثبات ان‌پی- سخت بودن مسئله ساده‌سازی تحت معیار مساحت قابلیت دید

مسئله دیگری که برای ساده‌سازی مسیر تحت معیار قابلیت دید بررسی می‌کنیم، مسئله ساده‌سازی برای مجموع خطای مسیر و تحت معیار مساحت قابلیت دید است. ما نشان می‌دهیم که برای یک مسیر چندضلعی داده شده $\mathcal{P}$ واقع در چندضلعی ساده‌ی $\mathcal{R}$، و یک عدد مثبت k و یک هزینه‌ی C، مسئله‌ی یافتن یک ساده‌سازی $\mathcal{Q}$ با حداکثر k پاره‌خط و مجموع خطای تقریب C، تحت معیار مساحت قابلیت دید ضعیف، ان‌پی-سخت است. ما این کار را با کاهشِ مسئله فوق به مسئله مجموعِ زیر مجموعه[۵] انجام می‌دهیم.

الگوریتم تقریبی برای مسئله ساده‌سازی تحت معیار مساحت قابلیت دید

اثبات ان‌پی-سخت بودن این مسئله نشان می‌دهد که ما امیدی به راه حل چند جمله‌ای برای این مسئله نمی‌توانیم داشته باشیم (حداقل تا زمانی که ثابت نشود P=NP است). بنابراین، ارایه راه حل تقریبی می‌تواند راهی برای مواجهه با این مشکل باشد. به این منظور، الگوریتمی تقریبی برای مسئله ساده‌سازی فوق ارایه می‌دهیم. ما نشان می‌دهیم که برای یک مسیر چندضلعی $\mathcal{P}$ واقع در چندضلعی ساده $\mathcal{R}$، یک مقدار صحیح k، یک پارامتر ϵ، و یک پارامتر σ، یک مسیر ساده شده $\mathcal{Q}$ تحت معیار مساحت قابلیت دید، با حداکثر k ضلع و با جمع خطاهای

[۵] Subset-sum

حداکثر $\sigma\mathcal{H}$ بزرگتر از پاسخ بهینه را می‌توان در زمان $O(n^۲k^۲/\sigma)$ با استفاده از $O(nk^۲/\sigma)$ فضا محاسبه نمود؛ که در آن $\mathcal{H}$ مساحت چندضلعی قابلیت دید $\mathcal{P}$ است.

۳.۱ طرح گزارش

در بخش اول این پایان نامه (همین بخش) مقدمه ارایه شده است. در فصل ۲، مفاهیم و تعاریف مسئله ارایه می‌شوند. از مفاهیم و تعاریفی که در این بخش ارایه می‌شود، در فصل‌های بعدی استفاده می‌شود. در فصل ۳، مروری بر کارهای قبلی داریم. در فصل ۴، به ارایه الگوریتم‌هایی برای ساده‌سازی مسیر تحت معیارهای مساحت می‌پردازیم. در فصل ۵، به ارایه الگوریتم‌هایی برای ساده‌سازی هم‌فضای مسیر می‌پردازیم. در فصل ۶، الگوریتم‌های ساده‌سازی تحت معیار قابلیت دید را ارایه می‌کنیم. در نهایت در فصل ۷ به جمع‌بندی می‌پردازیم. در این پژوهش علاوه بر مسئله‌ها و الگوریتم‌های فوق ما به مسئله ساده‌سازی تین (گونه دو و نیم بعدی مسئله ساده‌سازی) نیز پرداختیم. برای این مسئله الگوریتم موازی مبتنی بر پردازنده گرافیکی ارایه دادیم. برای حفظ یکپارچگی پایان نامه این نتیجه در پیوست آ آمده است.

فصل ۲

مفاهیم و تعاریف

۱.۲ تعریف ساده‌سازی مسیر

فرض کنید $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ مجموعه نقطه‌های مسیر چندضلعی ورودی باشد. همچنین فرض کنید مسیر ساده است و خود را قطع نمی‌کند. یک مسیر چندضلعی $\mathcal{Q} = \{q_1 = p_1, q_2, \ldots, q_b = p_n\}$ با $b \leq n$، را یک ساده‌سازی از مسیر $\mathcal{P}$ می‌نامیم. دو گونه کلی از این مسئله وجود دارد: گونه *محدود* و گونه *نامحدود*. در گونه محدود هر نقطه $q_i \in \mathcal{Q}$ باید عضوی از زیردنباله‌ی نقطه‌های $\mathcal{P}$ باشد. اما در گونه *نامحدود* این محدودیت وجود ندارد و نقطه‌ها می‌توانند دلخواه باشند. دو هدف بهینه‌سازی برای این مسئله تعریف می‌شود: (۱) $\#$-کمینه، که در آن هدف یافتن یک ساده‌سازی با کمترین تعداد نقطه‌ها و خطای حداکثر ϵ است، و (۲) ϵ-کمینه که در آن هدف یافتن ساده‌سازی با حداکثر k نقطه با کمترین خطای ساده‌سازی است. ما هر پاره‌خط q_iq_{i+1} را یک اتصال می‌نامیم. از آنجایی که انتخاب هر پاره‌خط p_ip_j مسیر را کوتاه‌تر می‌کند، ما آن را میانبر می‌نامیم. هر اتصال q_lq_{l+1} متعلق به $\mathcal{Q}$، در واقع یک میانبر p_ip_j $(i < j)$ در $\mathcal{P}$ را مشخص می‌کند. ما زیرمسیر از p_i به p_j را توسط $\mathcal{P}(i,j)$ نشان می‌دهیم. خطای یک ساده‌سازی $\mathcal{Q}$ تحت معیار خطا (یا تابع فاصله) F توسط $\mathrm{Err}_F(\mathcal{Q})$ نشان داده شده و این خطا برابر $\mathrm{Err}_F^{\max}(\mathcal{Q}) = \max_{i=1}^{k-1} \mathrm{Err}_F(q_iq_{i+1})$ یا $\mathrm{Err}_F^{\mathrm{sum}}(\mathcal{Q}) = \sum_{i=1}^{k-1} \mathrm{Err}_m(q_iq_{i+1})$، در نظر گرفته می‌شود و در آن q_iq_{i+1} ساده‌سازی زیرمسیر $\mathcal{P}(s,t) = \{p_s = q_i, p_{s+1}, \ldots, p_{t-1}, p_t = q_{i+1}\}$ است و $\mathrm{Err}_F(q_iq_{i+1})$ برابر مقدار خطای تقریب $\mathcal{P}(s,t)$ با q_iq_{i+1} تحت معیار خطای F است. برخی معیارهای خطای متداول در مسئله ساده‌سازی مسیر عبارت هستند از هاسدورف[1]، فرشه، و مساحت.

[1] Hausdorff

۲.۲ چارچوب ایمایی و ایری

اغلب الگوریتم‌های گونه محدود ساده‌سازی مسیر، مبتنی بر چارچوب ایمایی و ایری [۲۴، ۱۰] هستند. در این چارچوب برای خطای ϵ داده شده، یک گراف بدون وزن جهت دار $G_\epsilon(\mathcal{P})$ به صورتی که در ادامه می‌آید، ساخته می‌شود: $G_\epsilon = (V, E_\epsilon)$ که در آن $V = \{p_1, \ldots, p_n\}$ و

$$E_\epsilon = \{(p_i, p_j) | \mathrm{Err}_F(p_i p_j, \mathcal{P}(i,j)) \leq \epsilon\}$$

که $\mathrm{Err}_F(p_i p_j)$ عبارت است از خطای میانبر $p_i p_j$ تحت معیار خطای دلخواه F. به عبارت دیگر گره‌های این گراف از مجموعه نقطه‌های مسیر $\mathcal{P}$ ساخته می‌شود و لبه‌های[۲] آن کلیه لبه‌های ممکن با خطای کمتر از ϵ است. به این ترتیب، برای یافتن پاسخ مسئله، کافی است کوتاه‌ترین مسیر از p_1 به p_n محاسبه شود. بنابراین، مسئله‌ی اصلی در الگوریتم‌های مبتنی بر این چارچوب، محاسبه‌ی کارایِ خطای کلیه میانبرها خواهد بود.

۳.۲ انواع مسیرها

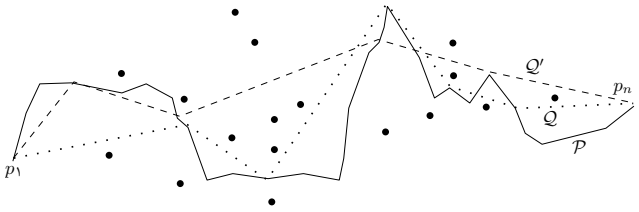
محققان الگوریتم‌های گوناگونی برای مسئله ساده‌سازی ارایه داده‌اند که بر روی انواع مختلفی از مسیرهای چندضلعی کار می‌کنند. در ادامه دو گونه از این مسیرها که در این پایان نامه به آن‌ها اشاره می‌کنیم را معرفی می‌کنیم.

- مسیرهای x-یکنوا: مسیری است که نسبت به یکی از محورهای مختصات یکنوا باشد. به عبارت دیگر، گوییم یک مسیر نسبت به محور xها یکنوا است، اگر از هر نقطه از محور xها خط عمودی رسم نماییم، مسیر را در بیش از یک نقطه قطع نکند. شکل ۱.۲آ نمونه‌ای از این مسیر را نشان می‌دهد.
- مسیرهای عمومی: این مسیر هیچ محدودیتی ندارد. تنها فرضی که وجود دارد این است که این مسیرها خود را قطع نمی‌کنند. شکل ۱.۲ب نمونه‌ای از این مسیر را نشان می‌دهد. چنین مسیرهایی را مسیر عمومی ساده یا به اختصار مسیر *ساده* می‌نامیم.

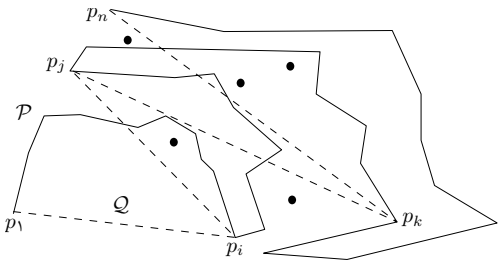
۴.۲ مسیرهای هم‌فضا

فرض کنید S مجموعه‌ای از m نقطه به عنوان مانع در صفحه باشند و فرض کنید این نقطه‌ها مسیر $\mathcal{P}$ را قطع نمی‌کنند. یک ساده‌سازی $\mathcal{Q}$ نسبت به مسیر $\mathcal{P}$ را هم‌فضا گویند اگر با فرض ثابت بودن نقطه‌های ابتدا و انتهای

[۲] Edges



(آ) مسیر ساده‌شده Q' هم‌فضایی را نسبت به مسیر اصلی $\mathcal{P}$ و مجموعه نقطه‌های S حفظ نمی‌کند. ساده‌سازی Q یک ساده‌سازی هم‌فضا بوده و ساده‌سازی معتبر محسوب می‌شود.



(ب) ساده‌سازی $\mathcal{Q}$ نسبت به $\mathcal{P}$ هم‌فضا است. ولی این ساده‌سازی قویا-هم‌فضا نیست، زیرا برخی میانبرهای $\mathcal{Q}$ مانند p_1p_i و p_jp_k به ترتیب نسبت به $\mathcal{P}(1,i)$ و $\mathcal{P}(j,k)$ هم‌فضا نیستند.

شکل ۱.۲: ساده‌سازی هم‌فضا و قویا-هم‌فضا .

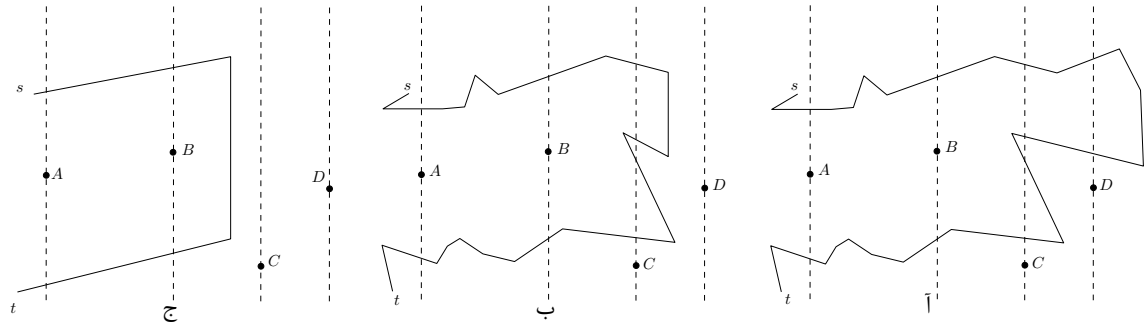
مسیر، بتوان به طور پیوسته و بدون عبور از روی نقطه‌های مجموعه S، مسیر $\mathcal{Q}$ را به مسیر $\mathcal{P}$ تبدیل نمود. به عبارت دقیق‌تر دو مسیر $\alpha, \beta : [0,1] \to R^2$ که دارای نقطه‌های ابتدایی و انتهایی مشترک هستند را نسبت به یکدیگر و با توجه به نقطه‌های مانع S هم‌فضا گویند اگر یک تابع پیوسته $\Gamma : [0,1] \times [0,1] \to R^2$ با خصوصیات زیر وجود داشته باشد:

۱. $\Gamma(0,t) = \alpha(t)$ و $\Gamma(1,t) = \beta(t)$ برای $0 \le t \le 1$،

۲. $\Gamma(s,0) = \alpha(0) = \beta(0)$ و $\Gamma(s,1) = \alpha(1) = \beta(1)$،

۳. $\Gamma(s,t) \notin S$ برای $0 \le s \le 1$ و $0 < t < 1$.

همچنین مسیر $\mathcal{P}$ و مسیر ساده‌شده آن $\mathcal{Q}$ را قویا-هم‌فضا گوییم اگر به ازای هر میانبر q_lq_{l+1} از $\mathcal{Q}$ که نماینده میانبر p_ip_j است، زیرمسیر $\mathcal{P}(i,j)$ و p_ip_j هم‌فضا باشند. از این پس، برای راحتی، چنین میانبری را *میانبر هم‌فضا* می‌نامیم. اگر دو مسیر $\mathcal{P}$ و $\mathcal{Q}$ قویا-هم‌فضا باشند آنگاه آن دو حتما هم‌فضا هستند ولی اگر دو مسیر هم‌فضا باشند، آن دو حتما قویا-هم‌فضا نیستند (شکل ۱.۲ب را ببینید). با این حال در حالتی که مسیر $\mathcal{P}$ x-یکنوا باشد می‌توان اثبات نمود اگر $\mathcal{P}$ و $\mathcal{Q}$ قویا-هم‌فضا باشند آنگاه آن دو حتما هم‌فضا هستند. به آسانی می‌توان عکس این قضیه را نیز برای مسیرهای x-یکنوا ثابت نمود. شکل ۱.۲آ نمونه‌ای از دو ساده‌سازی، یکی هم‌فضا (معتبر) و دیگری غیر-هم‌فضا رانشان می‌دهد. همچنین دو مسیر دارای یک کلاس یا رده‌ی هم‌فضایی هستند اگر آن دو مسیر نسبت به یکدیگر هم‌فضا باشند.



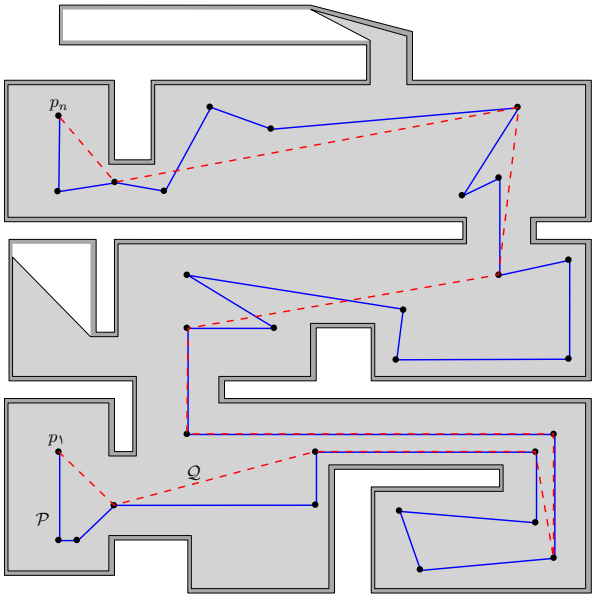
شکل ۲.۲: (آ) مسیر α با دنباله‌ی $\overline{ABCDDCCC}\underline{BA}$. (ب) تغییر شکل α که $\overline{DD}$ را حذف کرده است. (ج) مسیر تغییر شکل داده شده‌ی α با دنباله‌ی $\overline{AB}\underline{BA}$ که آنرا $\mathrm{CS}(\alpha)$ می‌نامیم.

۵.۲ دنباله‌ی فشرده‌ی یک مسیر($\mathrm{CS}(\alpha)$)

صفحه‌ای حاوی تعدادی مانع و مسیر α را در نظر بگیرید. فرض کنید هر مانع توسط یک نماد نام‌گذاری شده است. یک راه برای نمایش رده هم‌فضایی، نمایش مسیر α به صورت دنباله‌ای از نمادهای مانع‌هایی است که مسیر از بالا یا پایین آن‌ها می‌گذرد. به بیان دقیق‌تر فرض کنید یک خط عمودی از هر مانع گذر داده شده است. حال از نقطه‌ی آغازین s در طی مسیر α تا نقطه‌ی پایانی آن، t، حرکت می‌کنیم. در طی حرکت، هرگاه به یک خط عمودی گذرنده از یک مانع، مثلا A، می‌رسیم، اگر بالای A باشیم، نماد $\overline{A}$، و اگر پایین آن باشیم نماد $\underline{A}$ را در دنباله می‌نویسیم. به عنوان مثال، دنباله‌ی مسیر α در شکل ۲.۲آ برابر با $\overline{ABCDDCCC}\underline{BA}$ است. این دنباله را می‌توان کوتاه‌تر نمود بدون آن که رده هم‌فضایی آن تغییر کند. به این ترتیب که اگر دو نماد متوالی، یکسان باشند (برای مانع A، هر دو $\overline{A}$ یا هردو $\underline{A}$ باشند)، هردوی آن‌ها را می‌توان با تغییر شکل مسیر یا کوتاه نمودن آن، حذف نمود. شکل ۲.۲ب تغییر شکل، ناشی از حذف $\overline{DD}$ را از دنباله‌ی رشته مسیر نشان می‌دهد. این حذف را می‌توان تا زمانی که هیچ دو نماد یکسانی کنار هم نباشند ادامه داد. رشته‌ی به دست آمده را *دنباله‌ی فشرده* نامیده و با $\mathrm{CS}(\alpha)$ نشان می‌دهیم. شکل ۲.۲ج مسیر α را پس از انجام چند تغییر شکل و به دست آوردن $\mathrm{CS}(\alpha)$ نشان می‌دهد. در [۲۵] نشان داده شده است که دو مسیر α و β با نقطه‌های آغازین و پایانی یکسان، هم‌فضا هستند اگر و فقط اگر داشته باشیم $\mathrm{CS}(\alpha) = \mathrm{CS}(\beta)$. روشن است که CS برای یک مسیر، یکتا است. از این پس وقتی از عبارت *کوتاه‌کردن* یک مسیر استفاده می‌کنیم، منظور این است که مسیر را چندین بار تغییر شکل دهیم تا آن که دنباله کانونی آن را به دست آوریم.

۶.۲ قابلیت دید ضعیف

می‌گوییم یک نقطه مانند v توسط پاره خط e به صورت *ضعیف قابل دید* است اگر نقطه‌ای مانند $w \in e$ باشد به نحوی که v و w یکدیگر را ببینند. پاره‌خط e واقع در چندضلعی بسته $\mathcal{R}$ را در نظر بگیرید. *چندضلعی قابلیت*



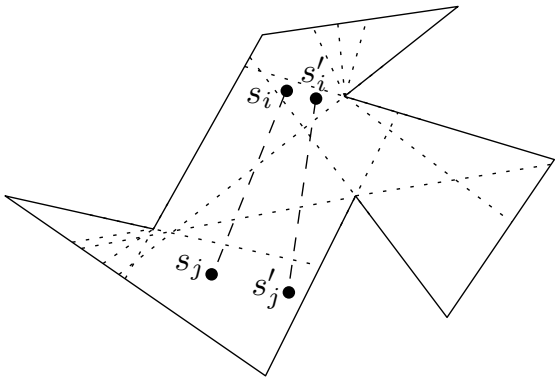
شکل ۲.۳: مسیر $\mathcal{P}$ واقع در چندضلعی ساده (نقشه ساختمان) و یک ساده‌سازی $\mathcal{Q}$ از آن (مسیر منقطع). ناحیه خاکستری، ناحیه‌ای است که هم توسط $\mathcal{P}$ و هم توسط $\mathcal{Q}$ به صورت ضعیف قابل دید است.

دید ضعیف برای e عبارت است از اجتماع همه نقطه‌هایی از چندضلعی $\mathcal{R}$ که به صورت ضعیف توسط e قابل دید هستند.

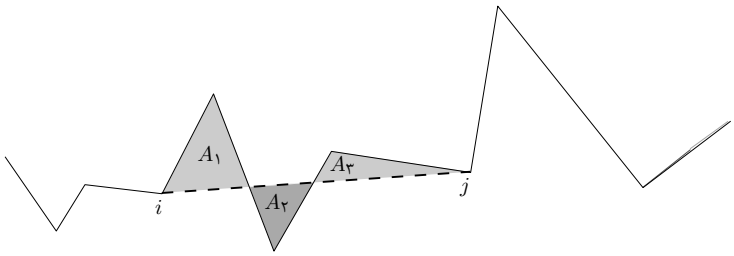
فرض کنید مسیر $\mathcal{P}$ واقع در چندضلعی $\mathcal{R}$ به ما داده شده است. چندضلعی قابلیت دید ضعیف $\mathcal{P}(i,j)$ عبارت خواهد بود از اجتماع چندضلعی‌های قابلیت دید توسط همه ضلع‌های p_ap_{a+1}، $i \leq a < j$. توجه داشته باشید که در این حالت یک نقطه/ضلع توسط یک زیرمسیر به صورت ضعیف قابل دید است اگر حداقل توسط یک نقطه از زیرمسیر قابل دید باشد.

۷.۲ بخش‌بندیِ ناحیه‌های قابلیت دید

فرض کنید $\mathcal{R}$ یک چند ضلعی ساده با اندازه m باشد. یک *بخش‌بندی از ناحیه‌های قابلیت دید* عبارت است از تقسیم $\mathcal{R}$ به مجموعه‌ای از ناحیه‌ها، به طوری که برای هر نقطه در داخل یک ناحیه، دنباله مشابهی از نقطه‌ها و ضلع‌های $\mathcal{R}$ قابل دید باشند (شکل ۴.۲ را ببینید). ثابت شده است که در یک چندضلعی ساده، تعداد چنین ناحیه‌هایی از مرتبه $O(m^3)$ است [۲۶].



شکل ۴.۲: چندضلعی بخش‌بندی شده به ناحیه‌های قابلیت دید



شکل ۵.۲: زیرمسیر $\mathcal{P}(i,j)$ توسط میانبر p_ip_j ساده شده است. ناحیه‌ی خاکستری روشن و خاکستری تیره، به ترتیب ناحیه‌های بالا و پایین واقع در بین زیرمسیر و میانبر متناظر را نشان می‌دهند.

۸.۲ معیار خطای مساحت

فرض کنید زیرمسیر $\mathcal{P}(i,j)$ توسط میانبر p_ip_j ساده شده باشد. همانگونه که در شکل ۵.۲ نشان داده شده است، در این ساده‌سازی ناحیه‌هایی هستند که در دو سمت میانبر p_ip_j توسط $\mathcal{P}(i,j)$ محیط می‌شوند. برای مثال آورده شده در شکل ۵.۲، این ناحیه‌ها عبارتند از $\{A_1, A_2, A_3\}$. استفاده از اجتماع مساحت‌های این ناحیه‌ها یک معیار طبیعی به عنوان خطای ساده‌سازی p_ip_j است. این معیار خطا را *جمع-مساحت* (*sum-area*) می‌نامند [۱۶، ۱۹، ۴]. در برخی کاربردها مانند ساده‌سازی مرزهای بین دو کشور همسایه، تفاوت بین مساحت ناحیه‌های مابین p_ip_j و $\mathcal{P}(i,j)$ در دو سمت p_ip_j مهم است. این معیار خطا *تفاضل-مساحت* (*diff-area*) نامیده می‌شود. مقدار این معیار برای میانبر p_ip_j در شکل ۵.۲ برابر است با $(\mathrm{Ar}(A_1)+\mathrm{Ar}(A_2))-\mathrm{Ar}(A_3)$ و در آن $\mathrm{Ar}(S)$ مساحت ناحیه‌ی بسته S است. به علاوه ما مقدار خطا برای یک میانبر تحت معیار جمع-مساحت و تفاضل-مساحت را به ترتیب با $\mathrm{Err_s}$ و $\mathrm{Err_d}$ نشان می‌دهیم.

معیارهای جمع-مساحت و تفاضل-مساحت، معیارهایی طبیعی در کاربردهای گرافیکی هستند. در چنین کاربردهایی مساحت ناحیه‌ی ساده شده باید تا آنجایی که ممکن است به مساحت شکل اصلی نزدیک باشد. این موضوع امری طبیعی است زیرا مساحت می‌تواند میزان شباهت یا میزان تفاوت بین دو ناحیه را تشخیص دهد.

۹.۲ معیار خطای قابلیت دید

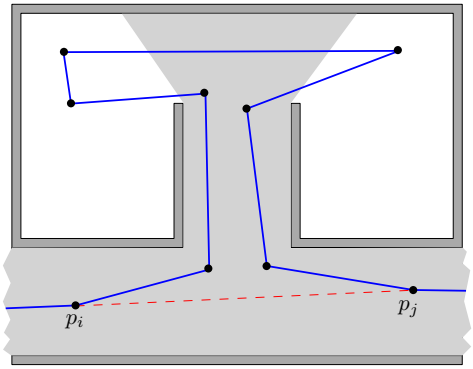
فرض کنید مسیر $\mathcal{P}$ واقع در چندضلعی $\mathcal{R}$ به ما داده شده است. ما دو معیار خطا برای محاسبه‌ی خطای یک میانبر p_ip_j از مسیر $\mathcal{P}$ ارایه می‌دهیم: (آ) WVS: عبارت است از اندازه‌ی تفاضل چند ضلعی قابلیت دید ضعیفِ میانبر p_ip_j و چندضلعی قابلیت دید ضعیفِ مسیر $\mathcal{P}(i,j)$ (منظور از اندازه، تعداد ضلع‌های چندضلعی است)، و (ب) WVA: عبارت است از تفاضل مساحت چندضلعی قابلیت دید ضعیف p_ip_j و مساحت چندضلعی قابلیت دید ضعیف زیرمسیر $\mathcal{P}(i,j)$. ما خطای یک میانبر را تحت این دو معیار به صورت زیر تعریف می‌کنیم:

$$\mathrm{Err}_{\mathrm{WVS}}(p_ip_j) = \mathrm{WVP}(\mathcal{P}(i,j)) \ominus \mathrm{WVP}(p_ip_j),$$

$$\mathrm{Err}_{\mathrm{WVA}}(p_ip_j) = \mathrm{Ar}(\mathrm{WVP}(\mathcal{P}(i,j)) \backslash \mathrm{WVP}(p_ip_j)),$$

که در آن $\mathrm{WVP}(.)$ چندضلعی قابلیت دید ضعیف یک میانبر یا یک زیرمسیر در داخل $\mathcal{R}$ است، $\mathrm{Ar}(X)$ مقدار مساحت چندضلعی X است، و $A \ominus B$ اندازه تفاضل بین چندضلعی‌های A و B است، به عبارتی $\mathrm{Err}_{\mathrm{WVS}}(p_ip_j)$ برابر است با تعداد ضلع‌هایی از چندضلعی که به صورت ضعیف توسط $\mathrm{WVP}(\mathcal{P}(i,j))$ قابل دیدن هستند ولی توسط $\mathrm{WVP}(p_ip_j)$ دیده نمی‌شوند (حتی به صورت ضعیف). شکل ۶.۲ مثالی برای این خطاها را نشان می‌دهد.

این معیار در جاهای مختلفی نظیر رباتیک کاربرد دارد. نقشه‌ی یک ساختمان و یک ربات نگهبان را در درون ساختمان در نظر بگیرید. به این ربات یک مسیر چندضلعی داده شده است و ربات وظیفه دارد روی این مسیر حرکت کند و در حین حرکت وجود هرگونه شیء ناشناسی را گزارش کند. قابلیت دید ربات را ۳۶۰ درجه در نظر بگیرید. می‌خواهیم مسیر کوتاه‌تر و کمینه‌ای را پیدا کنیم به طوری که قابلیت دید ربات به صورت محلی چندان تغییر نکند ($\mathrm{Err}^{\max}$) یا آن که کل قابلیت دید ربات (مجموع خطای مسیر) از میزان مشخصی بیشتر نشود ($\mathrm{Err}^{\mathrm{sum}}$). شکل ۳.۲ را ببینید که در آن قابلیت دید مسیر $\mathcal{P}$ و مسیر $\mathcal{Q}$ یکسان است.



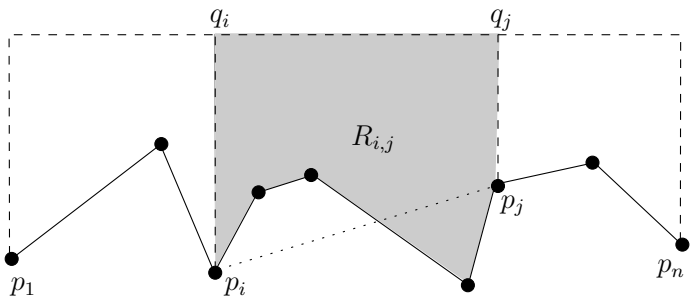
شکل ۲.۶: ناحیه خاکستری چندضلعی قابلیت دید ضعیف میانبر p_ip_j است. اجتماع ناحیه‌های خاکستری و سفید، ناحیه‌ای که به صورت ضعیف توسط زیرمسیر $\mathcal{P}(i,j)$ قابل دیدن است را نشان می‌دهد. خطایی که با گذر از میانبر p_ip_j ایجاد می‌شود، تحت معیار مساحت چندضلعی قابلیت دید ضعیف، مساحت ناحیه سفید در داخل چندضلعی است. همچنین $\mathrm{Err}_{\mathrm{WVS}}(p_ip_j)$ برابر با ۶ است (تعداد ضلع‌هایی از چندضلعی که توسط p_ip_j به صورت ضعیف دیده نمی‌شوند ولی توسط $\mathcal{P}(i,j)$ دیده می‌شوند).$\mathrm{Err}_{\mathrm{WVA}}(p_ip_j)$ برابر با مساحت ناحیه سفید است.

فصل ۳

کارهای مرتبط

۱.۳ ساده‌سازی مسیر

در این قسمت مروری بر کارهای مرتبط با پژوهش ما در زمینه گونه‌ی محدود ساده‌سازی مسیر داریم. برخی نتیجه‌ها برروی گونه‌ی نامحدود را می‌توان در منبع‌های [۲۷، ۲۲، ۲۴] مشاهده نمود. معروف‌ترین و یکی از قدیمی‌ترین روش‌ها برای ساده‌سازی مسیر توسط داگلاس و پکر ارایه شده است [۱]. این الگوریتم از فاصله عمودی به عنوان معیار خطا استفاده می‌کند. این الگوریتم را می‌توان در زمان $O(n^۲)$ پیاده‌سازی نمود. پیاده‌سازی‌هایی وجود دارند که زمان اجرای این روش را به $O(n \log n)$ [۲۸] و $O(n \log^* n)$ [۲۹] بهبود داده‌اند. با این حال، این الگوریتم یک الگوریتم مکاشفه‌ای است و تضمینی راجع به کیفیت ساده‌سازی ارایه نمی‌دهد. اولین چارچوب عمومی که پاسخ بهینه ساده‌سازی را برای گونه‌ی محدود مسئله ارایه می‌دهد توسط ایمایی و ایری ارایه شد [۱۰]. آن‌ها مسئله را با یک گراف جهت‌دار و بدون دور مدل کردند و نشان دادند که یک پاسخ بهینه را می‌توان با جستجوی کوتاه‌ترین مسیر در این گراف یافت. به علاوه آن‌ها نشان دادند که چگونه می‌توان این گراف را برای معیار خطای هاسدورف در زمان $O(n^۲ \log n)$ ساخت. زمان اجرای این الگوریتم به درجه‌ی دو یا نزدیک به درجه‌ی دو نیز بهبود یافته است [۱۳، ۱۱]. همچنین، برای معیار $L_۱$ و معیار خطای یکنواخت الگوریتمی با مرتبه‌ی زمانی $O(n^{۴/۳+\delta})$، برای هر $\delta > ۰$، ارایه شد [۱۲]. سرانجام، الگوریتم تقریبی با زمان نزدیک به خطی برای معیار خطای فاصله‌ی عمودی تحت متریک $L_۲$ ارایه شد [۱۵]. ساده‌سازی مسیر تحت معیار فرشه اولین بار در [۳۰] مطالعه شد. برای این معیار، بهترین نتیجه توسط نتیجه‌های [۱۰، ۳۱] قابل به دست آوردن است. برای نتیجه‌های بیشتر [۳۲، ۳۳، ۱۷] و مرجع‌های داخل آن‌ها را ببنید.



شکل ۱.۳: محاسبه‌ی تفاضل-مساحت برای مسیرهای x-یکنوا

۲.۳ ساده‌سازی مسیر تحت معیار مساحت

اولین الگوریتم ساده‌سازی تحت معیار جمع-مساحت توسط ورجین [۴] پیشنهاد شد. این روش خطای کلیه‌ی حالت‌های ساده‌سازی قابل تعریف بر روی نقطه‌های یک مسیر را محاسبه می‌کند [۴]. با این حال این الگوریتم نمایی بوده و در عمل کاربرد مناسبی ندارد. الگوریتم‌های مکاشفه‌ای یا حریصانه‌ی متعددی، مانند روشی که توسط ویزوالینگام و وایات ارایه شد [۱۹]، نیز وجود دارند. اشکال اصلی در این روش‌ها این است که آن‌ها هیچ تضمینی برای مقدار انحراف خروجی نسبت به پاسخ بهینه ارایه نمی‌دهند.

اخیرا بوز و همکارانش به مسئله ساده‌سازی تحت معیار مساحت پرداختند [۱۶] و الگوریتم‌های ساده‌سازی برای مسیرهای x-یکنوا ارایه نمودند. برای معیار خطای جمع-مساحت، آن‌ها ابتدا مسیر ورودی را توسط دو خط عمودی گذرنده از p_1 و p_n و یک خط افقی در بالای مسیر $\mathcal{P}$ محصور می‌کنند (شکل ۱.۳ را ببینید). سپس به صورت بازگشتی ناحیه‌ی محصور R را بخش‌بندی می‌نمایند و مساحت بالا و پایین $\mathcal{P}$ را برای هر زیرمسیر $\mathcal{P}(i,j)$ با استفاده از الگوریتم لنگرمن [۳۴] می‌سازند. برای یک چندضلعی ساده Z با n نقطه و m خط پرسمان، الگوریتم لنگرمن، مساحت Z را در دو سمت هر خط برای هر $\epsilon > 0$ در زمان $O(m^{2/3}n^{2/3+\epsilon}+(n+m)polylog\, n)$ حساب می‌کند. با استفاده از این روش، آن‌ها گراف جهت‌دار و بدون دور G را در زمان و حافظه‌ی $O(n^{2+\epsilon})$ محاسبه می‌کنند. برای معیار مساحت آن‌ها مجددا مسیر ورودی را به مانند شکل ۱.۳ محصور می‌کنند. $T_{i,j}$ را ذوزنقه‌ای که توسط p_i, p_j, q_j, q_i مشخص شده باشد در نظر بگیرید. روشن است که $\mathrm{Ar}(R_{i,j}) = \mathrm{Ar}(R_{1,j}) - \mathrm{Ar}(R_{1,i})$ بوده و $\mathrm{Err_d}(p_ip_j) = \mathrm{Ar}(R_{i,j} - \mathrm{Ar}(T_{i,j}))$ است. با استفاده از این روش و محاسبه‌ی مقدارهای $R_{1,j}$، مقدار $\mathrm{Err_d}(p_ip_j)$ را می‌توان در زمان $O(n^2)$ محاسبه نمود. روشن است که این روش‌ها تنها بر روی مسیرهای x-یکنوا کار می‌کنند و نمی‌توانند برای مسیرهای عمومی مورد استفاده قرار گیرند. بنابراین پرسش این است که آیا می‌توان این مسئله را برای مسیرهای عمومی و در زمان مناسب حل نمود.

۳.۳ ساده‌سازی هم‌فضای مسیر

الگوریتم‌های زیادی برای ساده‌سازی در حضور سایر شیءها به طوری که مفهوم هم‌فضایی رعایت شده باشد، وجود ندارد. اخیرا محققین مسایل هندسی به مسایل زیر علاقه نشان داده‌اند: تشخیص این که آیا دو مسیر هم‌فضا هستند یا خیر، [۲۵] محاسبه‌ی کوتاه‌ترین مسیر هم‌فضا [۳۵] برای یک مسیر داده شده، و تبدیل یک مسیر به دیگری با کمترین هزینه [۳۶، ۳۷].

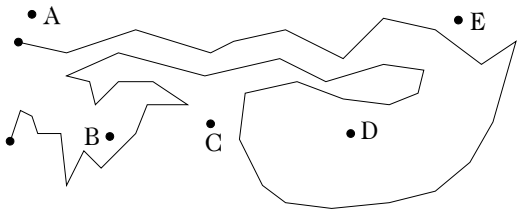
در حوزه‌ی ساده‌سازی مسیر، دی‌برگ و همکارانش [۲۰، ۲۱] اولین الگوریتم برای مسئله ساده‌سازی کمینه نقطه‌ها همراه با حفظ هم‌فضایی را ارایه نمودند. آن‌ها مسئله $\#$-کمینه را تحت معیار هاسدورف بررسی نمودند و الگوریتمی با پیچیدگی زمانی $O(n(n+m)\log n)$ ارایه نمودند. این الگوریتم، برای مسیرهای x-یکنوا ، مسیر هم‌فضا با کمینه تعداد نقطه‌ها را پیدا می‌کند. آن‌ها این الگوریتم را برای مسیرهای عمومی تعمیم دادند و یک روش مکاشفه‌ای برای این مسئله ارایه دادند. الگوریتم آن‌ها برای مسیرهای عمومی، کمینه بودن تعداد نقطه‌ها را تضمین نمی‌کند. گویباس و همکارانش [۲۲] نشان دادند که گونه نامحدود مسئله ساده‌سازی هم‌فضای بهینه برای حالتی که بخواهیم تضمین کنیم که خروجی خودش را قطع نکند، انپی-سخت است. همچنین، استوسکی و میتچل نشان دادند که ساده‌سازی هم‌فضای k مسیر، به طوری که یکدیگر را قطع نکنند، کمینه پی‌بی-کامل[۱] است و برای آن الگوریتم مکاشفه‌ای ارایه نمودند [۲۳].

۱.۳.۳ الگوریتم تشخیص هم‌فضا بودن دو مسیر

همان‌گونه که اشاره شده، برای تشخیص هم‌فضا بودن دو مسیر، کابلو و همکارانش الگوریتمی را ارایه نمودند [۲۵]. ما از ایده‌های این الگوریتم در یکی از الگوریتم‌هایی که ارایه می‌دهیم (بخش ۲.۱.۵)، استفاده می‌کنیم. در این جا به صورت خلاصه این الگوریتم را بیان می‌کنیم. این الگوریتم بررسی می‌کند که آیا دو مسیر ساده α و β، با اندازه‌ی n و با نقطه‌های ابتدایی و انتهایی مشترک که به همراه m مانع در صفحه قرار دارند، هم‌فضا هستند یا خیر. یک راه برای نمایش کلاس هم‌فضای یک مسیر مانند α این است که یک دنباله از مانع‌هاکه از بالا یا پایین آن می‌گذرند را بنویسیم (شکل۲.۳، ۳.۳ و ۴.۳ را ببینید). یک جفت همسایه تکرار شده از مانع‌ها را می‌توان با تغییر شکل دادن مسیر(بدون تغییر کلاس هم‌فضایی) حذف نمود. این حذف را تا آن جا که دنباله فشرده شده پیدا شود، می‌توان ادامه داد. یک مانع تکرار شده با علامت مخالف (بار)، یک مانع دوری شناخته می‌شود.

برای محاسبه کارای دنباله فشرده از یک مسیر α، الگوریتم به صورت زیر کار می‌کند. ابتدا مسیر α به

[۱] MIN PB-Complete: نشان داده شده است که مسئله‌هایی که در رده‌ی کمینه پی‌بی-کامل قرار دارند، قابل تقریب زدن با ضریب $n^{۱-\epsilon}$ برای $\epsilon > ۰$ نیستند. برای تعریف دقیق کمینه پی‌بی-کامل [۳۸] را ببینید.

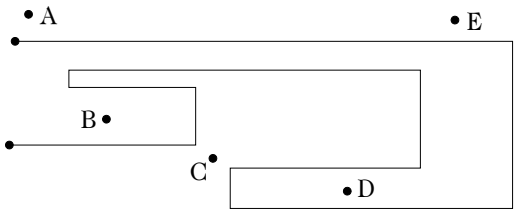


شکل ۲.۳: مسیر اصلی با دنباله $\underline{A}\overline{BCD}\underline{EED}\overline{DDCBB}\underline{BA}$.

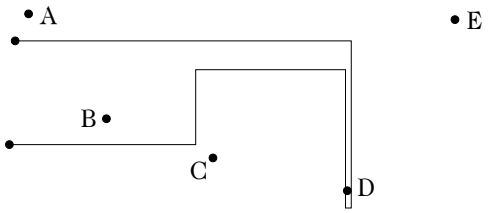
زیرمسیرهای x-یکنوا حداکثری (با حداکثر اندازه مناسب)، تجزیه می‌شود. مانع‌ها و زیرمسیرهای x-یکنوا با یکدیگر یک ترتیب یا رابطه‌ی بالایی-پایینی می‌سازند. این رابطه یک رابطه غیردوری است. در واقع این رابطه یک ترتیب جزئی است که می‌تواند به یک ترتیب کامل تبدیل شود. حال مسیر α می‌تواند به صورت اصلاح شده[۲]، معروف به *مسیر اصلاح شده* نمایش یابد. در چنین مسیری هر زیرمسیر x-یکنوای حداکثری به صورت یک پاره‌خط افقی با مقدار مختصات y برابر با رتبه آن زیرمسیر در ترتیب کامل نمایش می‌یابد. آنگاه مسیر اصلاح شده، کوتاه شده تا *مسیر اصلاح شده‌ی فشرده* (از این پس آن را با $\mathrm{CRP}(\alpha)$ نشان می‌دهیم)، که دنباله فشرده‌ی مسیر α را نشان می‌دهد، به دست آید. این کار با استفاده از داده‌ساختار پرسمان-بازه-عمودی که بر روی m مانع ایجاد شده، انجام می‌شود. به عبارت دقیق‌تر، هر سه پاره‌خط متوالی در مسیر اصلاح شده (پاره‌خط میانی به صورت عمودی و دوتای دیگر افقی)، به صورتی که در ادامه می‌آید پردازش می‌شود. ابتدا نزدیک‌ترین مانع به پاره‌خط میانی در زمان $O(\log m)$ با بکارگیری داده‌ساختار پرسمان-بازه-عمودی به دست می‌آید. سپس پاره‌خط میانی را به سمت نزدیک‌ترین مانع حرکت می‌دهیم به گونه‌ای که از نقطه‌های انتهایی دو پاره‌خط عمودی دیگر گذر نکند.

برای آزمایش این که مسیر α و β هم‌فضا هستند، الگوریتم ابتدا $\mathrm{CRP}(\alpha)$ و $\mathrm{CRP}(\beta)$ را به طریقی که گفته شد، محاسبه می‌کند. آنگاه بررسی می‌کند که آیا مانع‌های دوری α و β یک مجموعه را به دست می‌دهند یا خیر (توجه کنید که یک مانع دوری از α مانعی است که در آن $\mathrm{CRP}(\alpha)$ یک دور U شکل می‌زند). اگر این‌گونه نباشد، واضح است که α و β هم‌فضا نیستند. در غیر این صورت، این بدین معنی است که $\mathrm{CRP}(\alpha)$ و $\mathrm{CRP}(\beta)$ در یک یا چند مانع دوری شکسته شده و در نتیجه به زیرمسیرهای x-یکنوا تجزیه شده‌اند. آنگاه، هر زیرمسیر از $\mathrm{CRP}(\alpha)$ و زیرمسیر مربوطه به آن در $\mathrm{CRP}(\beta)$ برای بررسی هم‌فضا بودن، آزمایش می‌شوند. این آزمون‌ها با استفاده از خط جاروب و به صورت همزمان انجام شده و زیرمسیرهای x-یکنوا از $\mathrm{CRP}(\alpha)$ و $\mathrm{CRP}(\beta)$ از سمت چپ به راست بررسی می‌شوند.

[۲] Rectify



شکل ۳.۳: مسیر فشرده شده.



شکل ۴.۳: مسیر اصلاح شده با دنباله فشرده شده $\underline{A}\overline{BCD}\underline{D}\overline{C}\underline{BA}$.

۴.۳ ساده‌سازی تحت معیار قابلیت دید

گویباس و همکارانش [۳۹] نشان دادند که مسئله پیدا کردن چندضلعی قابلیت دید ضعیف برای یک پاره خط uv واقع در چندضلعی $\mathcal{R}$ را می‌توان در زمان $O(m)$ حل نمود. بایگی و قدسی [۴۰] نشان دادند که با انجام پیش پردازش از مرتبه زمانی $O(m^3 \log m)$ و حافظه $O(m^3)$، می‌توان چندضلعی قابلیت دید ضعیف برای یک پرسمان پاره‌خط را در زمان $O(\log m + k)$ پاسخ داد (k اندازه پاسخ و m اندازه چندضلعی $\mathcal{R}$ است). در زمینه ساده‌سازی تحت قابلیت دید، زارعی و قدسی [۴۱] الگوریتمی برای مسئله ساده‌سازی مرز یک چندضلعی تحت معیار خطای فاصله یک ناظر نقطه‌ای (واقع در چندضلعی) ارایه دادند. تا آن جایی که ما می‌دانیم، این تنها نتیجه مرتبط در زمینه ساده‌سازی تحت معیار قابلیت دید است.

فصل ۴

الگوریتم‌های ساده‌سازی مسیر مبتنی بر معیارهای مساحت

در این بخش به ارایه نتیجه‌های به دست آمده ما در زمینه‌ی ساده‌سازی مسیر تحت معیارهای مساحت پرداخته می‌شود. ما ابتدا یک تعریف یکپارچه برای معیارهای جمع-مساحت و تفاضل-مساحت به طوری که قابل اعمال بر روی مسیر عمومی باشد، ارایه می‌دهیم. الگوریتم ابتدایی برای ساده‌سازی بهینه تحت این دو معیار در زمان $O(n^۳)$ اجرا می‌شود، که در آن n اندازه‌ی مسیر است. برای معیار جمع-مساحت، یک الگوریتم تقریبی با مرتبه‌ی زمانی $O(n^۲/\epsilon)$ ارایه می‌دهیم، که در آن ϵ قدر مطلق مقدار خطای الگوریتم ما نسبت به حالت بهینه است. همچنین، برای خطای تفاضل-مساحت الگوریتمی ارایه می‌دهیم که در زمان $O(n^۲)$ ساده‌سازی بهینه را پیدا می‌کند. لازم به ذکر است که بهترین الگوریتم قبلی در این زمینه، تنها بر روی مسیرهای x-یکنوا کار می‌کند و الگوریتم‌های ما بر روی مسیرهای عمومی کار می‌کنند. الگوریتم‌های فوق با مشارکت آقای دکتر زارعی انجام شد. الگوریتم‌های ارایه شده ما در کنفرانس‌ها و ژورنال زیر چاپ شده‌اند:

1. S.Daneshpajouh, A.Zarei, M. Ghodsi, Path Simplification under Area Measures , Elsevier Graphical Models Journal, Volume 74, Issue 5, September 2012, Pages 283–289.

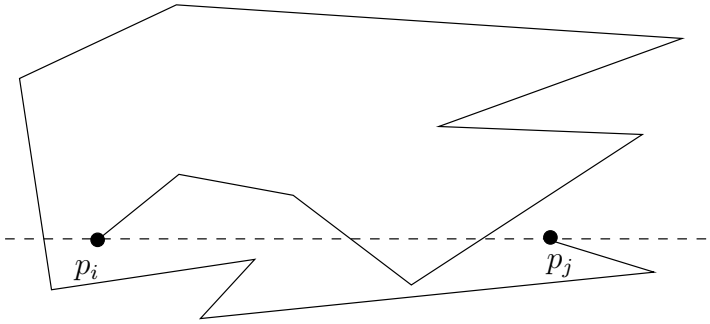
2. S. Daneshpajouh, A.Zarei, M. Ghodsi, Path Simplification under Difference Area Measure, 14th Int'l CSI Computer Conference (CSICC'2009), Amirkabri University of Technology, Pages 276–279, July 1-2, 2009, Tehran, Iran, 2009.

3. S. Daneshpajouh, A.Zarei, M. Ghodsi, On Realistic Line Simplification under Area Measure, In proceeding of 2009 IAENG International Conference on Computer Science (ICCS-2009), Hong Kong, Pages 465–470, 18–20 March, 2009.

فرض کنید $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ مجموعه نقطه‌های مسیر چندضلعی ورودی باشد. همچنین فرض کنید مسیر ساده است و خود را قطع نمی‌کند. یک مسیر چندضلعی $\mathcal{Q} = \{q_1 = p_1, q_2, \ldots, q_b = p_n\}$ با $b < n$، را یک ساده‌سازی از مسیر $\mathcal{P}$ در نظر می‌گیریم. در این فصل ما به گونه‌ی محدود مسئله ساده‌سازی مسیر که در آن مجموعه نقطه‌های $\mathcal{Q}$ زیرمجموعه‌ای از مجموعه نقطه‌های $\mathcal{P}$ هستند می‌پردازیم. ما الگوریتم‌هایی برای مسئله $\#$- کمینه تحت معیارهای جمع- مساحت و تفاضل- مساحت با بیشینه‌ی خطای مسیر، $\max_{i=1}^{k-1} \mathrm{Err}_F(q_i q_{i+1})$، ارایه می‌دهیم.

در تلاش برای استفاده از معیارهای جمع- مساحت و تفاضل- مساحت برای ساده‌سازی مسیر، ما به تعریف‌های مختلفی برای این معیارها برخورد کردیم که هیچکدام از آن‌ها برای یک مسیر عمومی به خوبی قابل اعمال نبودند. بنابراین ما ابتدا یک *تعریف یکپارچه* برای معیارهای جمع- مساحت و تفاضل- مساحت ارایه می‌دهیم به طوری که قابل اعمال بر روی مسیرهای عمومی باشد. برای شروع، نشان می‌دهیم که مسئله‌ی ساده‌سازی $\#$- کمینه تحت معیارهای جمع- مساحت و تفاضل- مساحت، با استفاده از تعریف یکپارچه ارایه شده ما، با به کارگیری یک روش ابتدایی در زمان $O(n^3)$ قابل حل است. این زمان اجرا برای کاربردهای واقعی بالا است. ما این مشکل را با ارایه الگوریتم‌هایی از درجه‌ی دو یا نزدیک به درجه‌ی دو با استفاده از تعریف یکپارچه‌ای که برای جمع- مساحت و تفاضل- مساحت ارایه می‌دهیم، حل می‌کنیم. به عبارت دقیق‌تر الگوریتم ما دارای مرتبه‌ی زمانی $O(n^2/\epsilon)$ بوده و مسیری با خطای ساده‌سازی حداکثر ϵL^2 نسبت به مسیر ساده‌سازی بهینه به دست می‌آورد، که در آن L طول بزرگترین میانبر ساده‌سازی است. ما فرض می‌کنیم برای یک کاربرد واقعی، مسیر و نقطه‌های آن در یک محدوده با طول ثابت جای می‌گیرند. بنابراین طول بزرگترین اتصال از هر ساده‌سازی یک مقدار ثابت بوده و این بدان معنی است که L با یک مقدار ثابت محدود شده است. پس این الگوریتم، که دارای زمان اجرای نزدیک به درجه‌ی دو است، می‌تواند برای یافتن تقریب‌هایی تا اندازه‌ی دلخواه نزدیک به ساده‌سازی بهینه مورد استفاده قرار گیرد. برای معیار خطای تفاضل- مساحت ما الگوریتمی پیشنهاد می‌دهیم که خطای کلیه میانبرها را در زمان $O(n^2)$ محاسبه می‌کند. در نتیجه ما یک الگوریتم ساده‌سازی بهینه با زمان اجرای $O(n^2)$ و خطای $\max_{i=1}^{k-1} \mathrm{Err}_{\mathrm{d}}(q_i q_{i+1})$ خواهیم داشت. تا آن جایی که ما می‌دانیم، این الگوریتم‌ها بهترین الگوریتم‌هایی هستند که می‌توانند برای ساده‌سازی مسیرهای عمومی تحت معیارهای جمع- مساحت و تفاضل- مساحت به کار گرفته شوند.

در بخش ۱.۴ ابتدا یک روش یکپارچه برای محاسبه‌ی مساحت بین یک زیرمسیر و میانبر آن ارایه می‌دهیم که از آن در محاسبه‌ی معیارهای جمع- مساحت و تفاضل- مساحت استفاده می‌کنیم. سپس الگوریتم‌ساده‌سازی بهینه را برای این معیارها ارایه می‌دهیم. در بخش ۲.۱.۴ الگوریتم تقریبی نزدیک به درجه‌ی دو برای ساده‌سازی مسیرهای عمومی تحت معیار جمع- مساحت را ارایه می‌دهیم. در بخش ۳.۴ الگوریتم ساده‌سازی بهینه با مرتبه‌ی زمانی درجه‌ی دو را ارایه می‌دهیم. در انتها نیز جمع‌بندی ارایه می‌شود.



شکل ۱.۴: یک زیرمسیر پیچیده که توسط میانبر p_ip_j ساده شده است.

۱.۴ ساده‌سازی بهینه تحت معیارهای مساحت

در این بخش، ما ابتدا تعریف مساحتِ بین یک زیرمسیر و میانبر متناظر آن را بازبینی نموده و یک معیار مساحت یکپارچه که هر نوع مسیری را بپوشاند ارایه می‌دهیم. ما نشان می‌دهیم که الگوریتم‌های فعلی ساده‌سازی را می‌توان بر روی این تعریف از جمع-مساحت و تفاضل-مساحت اعمال نمود و به بیان الگوریتم ابتدایی بهینه تحت این دو معیار می‌پردازیم.

۱.۱.۴ بازتعریف معیار مجموع مساحت

فرض کنید که زیرمسیر $\mathcal{P}(i,j) = \{p_i, p_{i+1}, \ldots, p_j\}$ توسط میانبر p_ip_j ساده شده است. خطای ساده‌سازی تحت معیار مساحت بستگی به مساحت ناحیه‌ی محدود شده توسط $\mathcal{P}(P(i,j)$ و p_ip_j دارد. در حالت کلی، زیرمسیر $\mathcal{P}(i,j)$ ممکن است با p_ip_j برخورد داشته باشد. تشخیص و محاسبه‌ی مساحت ناحیه محدود شده ممکن است بسیار پیچیده باشد. به عنوان مثال مسیر پیچیده‌ی نشان داده شده در شکل ۱.۴ را ببینید. بنابراین ما نیاز به تعریفی داریم که همه انواع مسیرها را بپوشاند.

ما ابتدا بین مساحت واقع در سمت چپ و سمت راست یک میانبر تفاوت قایل می‌شویم. اگر ما در نقطه‌ی p_i باشیم و به نقطه‌ی p_j نگاه کنیم، بخش‌هایی از زیرمسیر در سمت چپ واقع شده و سایر بخش‌ها در سمت راست قرار می‌گیرند. بنابراین ما دو مقدار داریم که مساحت ناحیه‌ی محدود شده توسط p_ip_j و $\mathcal{P}(i,j)$ را تعریف می‌کند: مساحت چپ و مساحت راست که به ترتیب با Ar_l و Ar_r نشان می‌دهیم. سپس، خطای یک میانبر p_ip_j تحت معیارهای جمع-مساحت و تفاضل-مساحت به ترتیب با $\mathrm{Err_s}(p_ip_j) = |\mathrm{Ar}_l(p_ip_j) + \mathrm{Ar}_r(p_ip_j)|$ و $\mathrm{Err_d}(p_ip_j) = |\mathrm{Ar}_l(p_ip_j) - \mathrm{Ar}_r(p_ip_j)|$ تعریف می‌کنیم که در آن $|.|$ قدرمطلق مقدار را نشان می‌دهد.

حالا ما نحوه‌ی محاسبه‌ی Ar_l و Ar_r را شرح می‌دهیم. فرض کنید $\mathcal{P}(i,j)$ میانبر p_ip_j را در نقطه‌های $\{p_{I1}, p_{I2}, \ldots, p_{Ik-1}, p_{Ik}\}$ قطع کند. در این حالت p_ip_j به پاره‌خط‌های $U = \{p_ip_{I1}, p_{I1}p_{I2}, \ldots, p_{Ik-1}p_k, p_kp_j\}$ تقسیم می‌شود. ما *خط حامی* p_ip_j را خط گذرنده از p_ip_j تعریف می کنیم. توجه کنید که ما، بین نقطه‌های

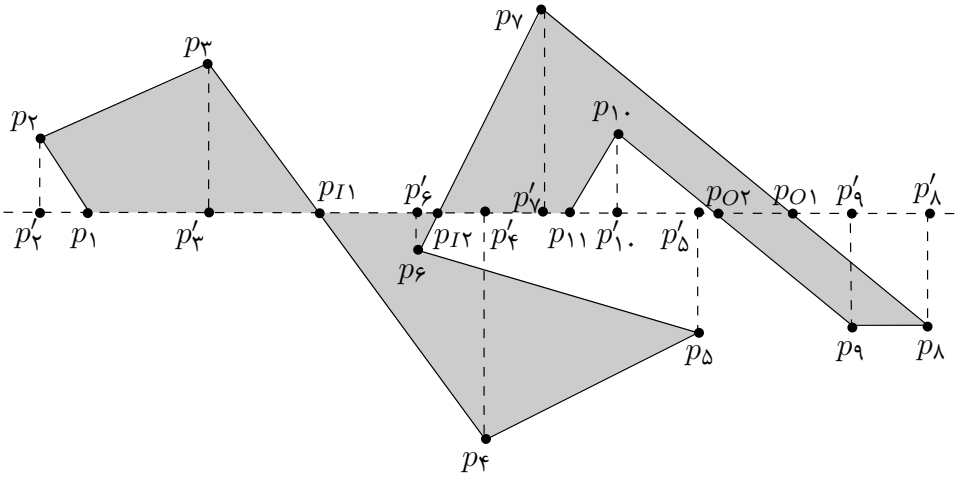
تقاطع $\mathcal{P}(i,j)$ و p_ip_j و نقطه‌های تقاطع $\mathcal{P}(i,j)$ و خط حامی p_ip_j تفاوت قایل می‌شویم. هر پاره‌خط $p_{Ix}p_{Iy} \in U$ و زیرمسیر متناظر آن، $\mathcal{P}(Ix,Iy)$، یک چندضلعی بسته را تعریف می‌کند. این چندضلعی ساده است زیرا با توجه به این که p_{Ix} و p_{Iy} دو نقطه متوالی در U هستند، $\mathcal{P}(Ix,Iy)$ نمی‌تواند خودش را قطع کند. ما این چند ضلعی را با $\Delta(Ix,Iy)$ نشان می‌دهیم. می‌گوییم که $\Delta(Ix,Iy)$ در سمت چپ(راست) p_ip_j واقع شده است اگر هر ضلع $\Delta(Ix,Iy)$ را بتوان توسط یک مسیر به سمت چپ (راست) $p_{Ix}p_{Iy}$ وصل نمود. برای محاسبه مقدارهای $\mathrm{Ar}_l(p_ip_j)$ و $\mathrm{Ar}_r(p_ip_j)$، ما مساحت هر یک از $\Delta(Ix,Iy)$ها را برای همه $p_{Ix}p_{Iy} \in U$، با توجه به سمت آنها، محاسبه می‌کنیم. ما این مقدارها را به $\mathrm{Ar}_l(p_ip_j)$ یا $\mathrm{Ar}_r(p_ip_j)$ اضافه می‌کنیم. مساحت هر یک از $\Delta(Ix,Iy)$، برای هر یک از $p_{Ix}p_{Iy} \in U$ به صورت زیر تعریف می‌شود:

$$\sum_{p_sp_t \in \mathcal{P}(Ix,Iy)} \text{Signed-Area}(p'_sp_sp_tp'_t)$$

که در آن p_sp_t یک ضلع از $\mathcal{P}(Ix,Iy)$، $x \le s < t \le y$ است و p'_s و p'_t به ترتیب تصویر عمود نقطه‌های p_s و p_t بر روی خط حامی $p_{Ix}p_{Iy}$ است. اگر p_sp_t خط حامی $p_{Ix}p_{Iy}$ را قطع کند، آنگاه هر قسمت از این ضلع به صورت جداگانه بررسی می‌شود. مقدار قدر مطلق $\text{Signed-Area}(p'_sp_sp_tp'_t)$ برابر با مساحت ذوزنقه $p'_sp_sp_tp'_t$ بوده و علامت آن به صورت زیر معین می‌شود:

- اگر $\overrightarrow{p'_sp'_t}$ و $\overrightarrow{p_{Ix}p_{Iy}}$ دارای جهت یکسان باشند و p_sp_t و $\Delta(Ix,Iy)$ در یک سمت $p_{Ix}p_{Iy}$ باشند آنگاه علامت $(\text{Signed-Area}(p'_sp_sp_tp'_t))$ مثبت است،
- در غیر این صورت علامت $(\text{Signed-Area}(p'_sp_sp_tp'_t))$ منفی است.

با استقرا بر روی طول زیرمسیر $\mathcal{P}(Ix,Iy)$ می‌توان به آسانی نشان داد که روش بالا با مساحت چندضلعی ساده $\Delta(Ix,Iy)$ برابر است. توجه داشته باشید که، اگر جهت ضلع‌های $\mathcal{P}(Ix,Iy)$ با جهت ضلع‌های $\mathcal{P}(i,j)$ یکسان نباشد، علامت مساحت محاسبه شده منفی است. به عنوان یک مثال، ساده‌سازی $p_۱p_{۱۱}$ از زیرمسیر $\mathcal{P}(۱,۱۱)$ را در شکل ۲.۴ در نظر بگیرید. مساحت چندضلعی $p_ip_{i+۱}\dots p_j$ توسط $\mathrm{Ar}(p_ip_{i+۱}\dots p_j)$ مشخص شده است.



شکل ۲.۴: ناحیه محصور شده توسط $\mathcal{P}(۱, ۱۱)$ و میانبر $p_۱p_{۱۱}$.

$$
\begin{aligned}
\mathrm{Ar}_l(p_۱p_{۱۱}) &= \mathrm{Ar}(\Delta(۱, I۱)) + \mathrm{Ar}(\Delta(I۲, ۱۱)) \\
&= (-\mathrm{Ar}(p_۱p_۲p'_۲) + \mathrm{Ar}(p'_۲p_۲p_۳p'_۳) + \\
&\quad \mathrm{Ar}(p'_۳p_۳p_{I۱})) + (\mathrm{Ar}(p_{I۲}p_۷p'_۷) + \\
&\quad \mathrm{Ar}(p'_۷p_۷p_{O۱}) - \mathrm{Ar}(p_{O۱}p'_۸p_۸) \\
&\quad +\mathrm{Ar}(p_۸p'_۸p'_۹p_۹) + \mathrm{Ar}(p'_۹p_۹p_{O۲}) \\
&\quad -\mathrm{Ar}(p_{O۲}p_{۱۰}p'_{۱۰}) - \mathrm{Ar}(p'_{۱۰}p_{۱۰}p_{۱۱})),
\end{aligned}
$$

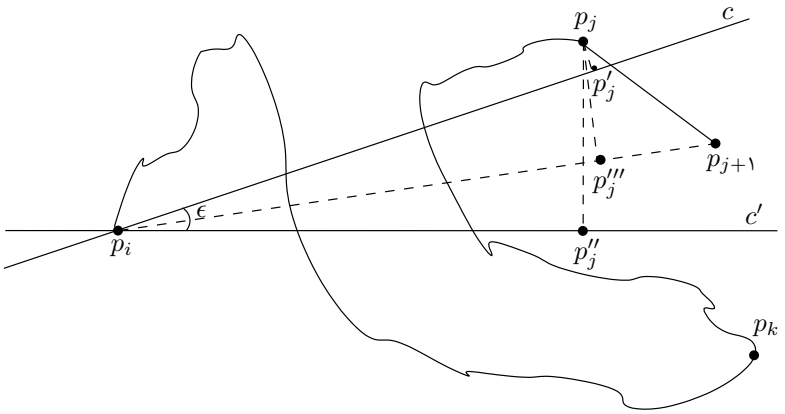
و

$$
\begin{aligned}
\mathrm{Ar}_r(p_۱p_{۱۱}) &= \mathrm{Ar}(\Delta(I۱, I۲)) \\
&= \mathrm{Ar}(u_۱p_۴p'_۴) + \mathrm{Ar}(p'_۴p_۴p_۵p'_۵) \\
&\quad -\mathrm{Ar}(p'_۵p_۵p_۶p'_۶) + \mathrm{Ar}(p_۶p'_۶p_{I۲})
\end{aligned}
$$

به آسانی می‌توان نشان داد که این محاسبه برابر مساحت ناحیه خاکستری رنگ در شکل ۲.۴ است.

۲.۱.۴ الگوریتم بهینه

از تعریف معیار مساحت که در بخش ۱.۱.۴ ارایه شد، می‌توان به عنوان یک تعریف یکپارچه و عمومی استفاده نمود و آن را بر روی هر مسیری اعمال نمود. در این بخش ما یک الگوریتم ابتدایی برای محاسبه معیارهای



شکل ۳.۴: تقریب‌زدن خطای یک میانبر.

خطای ارایه شده بیان می‌کنیم. سپس از چارچوب ایمایی و ایری [۱۰] استفاده نموده و به حل مسئله‌ی #-کمینه می‌پردازیم. ابتدا ما $\mathrm{Err_s}(p_ip_j)$ یا $\mathrm{Err_d}(p_ip_j)$ را با استفاده از یک پیمایش خطی روی زیرمسیر $\mathcal{P}(i,j)$ در زمان $O(j-i)$ محاسبه می‌کنیم. تعداد میانبرها از درجه $O(n^2)$ است. بنابراین خطای این تعداد میانبر را تحت یکی از این دو معیار خطا، و برای همه‌ی میانبرهای p_ip_j، در زمان $O(n^3)$ می‌توان محاسبه نمود. آنگاه، میانبرهای دارای خطای کمتر از ϵ به گراف جهت‌دار و بدون وزن G_ϵ اضافه شده و سپس کوتاه‌ترین مسیر را از p_1 به p_n محاسبه نموده و مسیر بهینه را پیدا می‌کنیم. بنابراین قضیه زیر را خواهیم داشت:

قضیه ۱.۱.۴. *الگوریتمی وجود دارد که مسئله ساده‌سازی #-کمینه تحت معیارهای خطای جمع-مساحت و تفاضل-مساحت را در زمان $O(n^3)$ و حافظه‌ی $O(n^2)$ محاسبه کند.*

۲.۴ الگوریتم تقریبی برای ساده‌سازی تحت معیار خطای جمع-مساحت

پیچیدگی زمانی الگوریتم بهینه، $O(n^3)$، برای استفاده در کاربردهای عملی مناسب نیست. در این بخش ما الگوریتمی با پیچیدگی زمانی نزدیک به درجه دو برای مسئله ساده‌سازی #-کمینه ارایه می‌دهیم که بر روی مسیرهای عمومی کار می‌کند. هرچند در این روش، ساده‌سازی به دست آمده بهینه نیست.

ایده الگوریتم تقریبی این است که از نتیجه محاسبه‌ی $\mathrm{Err_s}(p_ip_j)$ برای محاسبه‌ی کارای $\mathrm{Err_s}(p_ip_{j+1})$ استفاده شود. این کار را با استفاده از محاسبه و نگهداری خطای زیرمسیر فعلی برای مجموعه‌ای از خط‌های کانونی که از گره آغازین (در اینجا p_i) منشعب می‌شوند، انجام می‌دهیم. برای نقطه بعدی p_{j+1}، ما دو خط کانونی که p_{j+1} بین آن‌ها است (از این پس این دو خط را c و c' می‌نامیم) را پیدا نموده و خطای p_ip_{j+1} را با استفاده از خطای این دو خط محاسبه می‌کنیم.

فرض کنید که برای یک زیرمسیر $\mathcal{P}(i,j)$، ما مقدار دقیق $\mathrm{Ar}_l(p_ip'_j)$، $\mathrm{Ar}_r(p_ip'_j)$، $\mathrm{Ar}_l(p_ip''_j)$ و $\mathrm{Ar}_r(p_ip''_j)$

که در آن p'_j و p''_j به ترتیب تصویرهای عمود p_j بر روی خط‌های c و c' است، را داریم (شکل ۳.۴ را ببینید). برای نقطه بعدی p_{j+1}، ما از $|\mathrm{Ar}_l(p_ip'_j) + \mathrm{Ar}_r(p_ip''_j) + \text{Signed-Area}^*(p_jp'''_jp_{j+1})|$ به عنوان تقریبی برای $\mathrm{Err_s}(p_ip_{j+1})$ استفاده می‌کنیم، که در آن c سمت چپ p_ip_{j+1} واقع شده، c' سمت راست جای گرفته، p'''_j تصویر عمود p_j روی خط حامی p_ip_{j+1} بوده و $\text{Signed-Area}^*(p_jp'''_jp_{j+1})$ برابر مساحت علامت‌دار مثلث $p_jp'''_jp_{j+1}$ است. توجه داشته باشید که علامت Signed-Area^* مثبت است اگر و فقط اگر $\overrightarrow{p_ip_{j+1}}$ و $\overrightarrow{p'''_jp_{j+1}}$ هم جهت باشند.

ما به صورت افزایشی مساحت بالا و پایین هر خط کانونی $c \in C$ را حساب می‌کنیم. به عبارت دقیق‌تر، برای هر $c \in C$ و هر p_jp_{j+1}، ما یک مساحت علامت‌دار برای ذوزنقه $p_jp_{j+1}p'_{j+1}p'_j$، که آن را با $\nabla(c,i,j)$ نشان می‌دهیم، محاسبه می‌کنیم. در عبارت فوق p'_j و p'_{j+1} به ترتیب تصویرهای p_j و p_{j+1} روی c هستند. علامت این ناحیه مثبت است اگر و فقط اگر $\overrightarrow{p_jp_{j+1}}$ و $\overrightarrow{p'_jp'_{j+1}}$ دارای جهت یکسان باشند. به آسانی می‌توان نشان داد که $\nabla(c,s,t)$ برابر مساحت محاسبه شده با استفاده از روش یکپارچه ما، $\sum_{p_sp_t\in\mathcal{P}(i,j)} \text{Signed-Area}(p'_sp_sp_tp'_t)$، است. ما این مقدار تقریب زده شده را با $\mathrm{Err}^*_\mathrm{s}(p_ip_{j+1})$ نشان می‌دهیم.

لم ۱.۲.۴. *ما رابطه‌ی زیر را بین تقریب و مقدار دقیق جمع- مساحت یک ضلع p_ip_{i+1} داریم.*

$$\mathrm{Err_s}(p_ip_{j+1}) - \frac{\epsilon|p_ip_k|^2}{2} \leq \mathrm{Err}^*_\mathrm{s}(p_ip_{j+1})$$

و

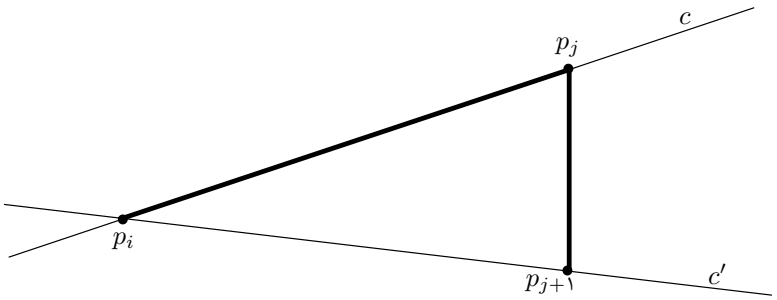
$$\mathrm{Err_s}(p_ip_{j+1}) \leq \mathrm{Err_s}(p_ip_{j+1}) + \frac{\epsilon|p_ip_k|^2}{2}$$

که در آن ϵ زاویه بین c و c' است و شامل نقطه p_{j+1} بوده، و p_k دورترین نقطه $\mathcal{P}$ از p_i است.

اثبات. با فرض این که ϵ به اندازه کافی کوچک است، ما می‌توانیم نتیجه بگیریم که $\mathrm{Ar}_l(p_ip'_j)-\epsilon|p_ip_k|^2/2 \leq \mathrm{Ar}_l(p_ip_{j+1})$ و $\mathrm{Ar}_r(p_ip_{j+1}) \leq \mathrm{Ar}_r(p_ip''_j) - \epsilon|p_ip_k|^2/2$.

این تفاوت مربوط به مساحتی است که بین c و c' قرار می‌گیرد. طبق تعریف p_k، وقتی ϵ به اندازه کافی کوچک باشد، این ناحیه حداکثر برابر $\epsilon|p_ip_k|^2/2$ است. شکل ۴.۴ نمونه‌ای از بدترین حالت ممکن را نشان می‌دهد. □

بنابراین، اگر ما برای مقدار کوچک ϵ، این خط‌های کانونی را داشته‌باشیم، می‌توانیم برای نقطه بعدی، با استفاده از خطاهای چپ و راست از مسیر $\mathcal{P}(i,j)$ بر روی این خط‌های کانونی، در زمان ثابت خطای جمع-مساحت را محاسبه کنیم. با این حال ما باید برای نقطه‌های جدید، ناحیه‌های بالا و پایین این خط‌های کانونی را بروز کنیم تا بتوانیم خطای نقطه بعدی را تقریب بزنیم.



شکل ۴.۴: نمونه‌ای از بدترین حالت برای تقریب $\mathrm{Err_s}(p_ip_{j+1})$.

لم ۲.۲.۴. *یک الگوریتم با زمان $O(\frac{2\pi}{\epsilon}n)$ وجود دارد که می‌تواند خطای جمع مساحت را برای کلیه میانبرهای p_1p_i، $1 < i \leq n$، به صورت تقریبی محاسبه کند.*

اثبات. $2\pi/\epsilon$ خط کانونی داریم که از p_1 خارج می‌شوند. به هنگام دریافت نقطه جدید مقدارهای Ar_l و Ar_r برای این خط‌ها بروزرسانی می‌شوند. آنگاه خطای تقریب‌زده شده‌ی میانبر جدید، در زمان ثابت محاسبه می‌شود. □

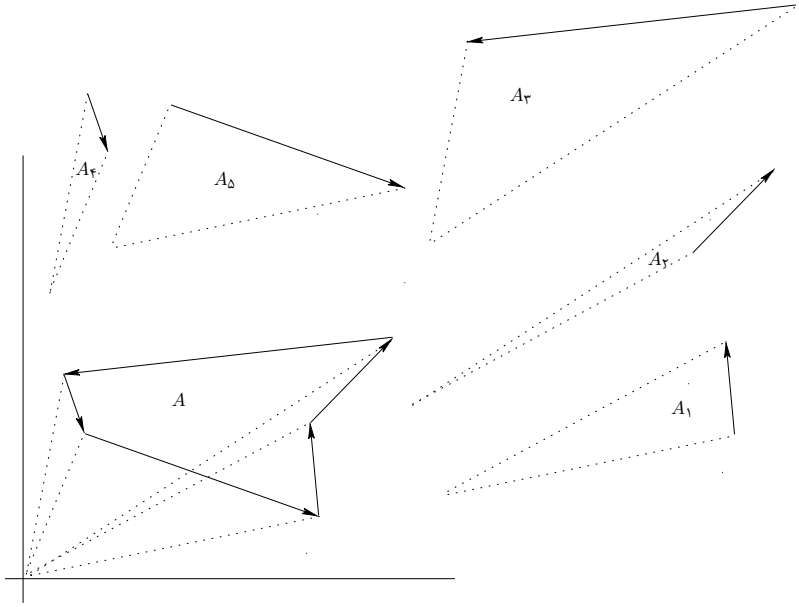
برای هر نقطه آغازین p_i ما می‌توانیم روش فوق را به کار گیریم. بنابراین، ما می‌توانیم خطای تقریب‌زده شده‌ی کلیه میانبرهای p_ip_j را در زمان $2\pi n^2/\epsilon$ به دست آوریم. همان‌گونه که قبلا اشاره شد، در یک سناریوی واقعی ما در یک ناحیه محدود کار می‌کنیم. آن‌گاه فاصله بین هر دو نقطه کمتر از یک مقدار ثابت خواهد بود. بنابراین در لم ۱.۲.۴ می‌توانیم از اندازه p_ip_k چشم‌پوشی کنیم. جمع‌بندی این نتیجه‌ها را در قضیه‌ی زیر خلاصه می‌کنیم.

قضیه ۳.۲.۴. *یک الگوریتم با پیچیدگی زمانی $O(n^2/\epsilon)$ وجود دارد که می‌تواند برای پیداکردن یک تقریب برای مسئله ساده‌سازی $\#$-کمینه برای بیشینه خطای مسیر به کار گرفته شود. در یک سناریوی واقعی خطای ساده‌سازی به اندازه‌ی $O(\epsilon)$ نسبت به خطای ساده‌سازی بهینه اختلاف دارد.*

۳.۴ الگوریتم ساده‌سازی مسیر مبتنی بر معیار تفاضل-مساحت

در این بخش ما الگوریتم کارایی را برای محاسبه‌ی خطای کلیه میانبرها تحت معیار تفاضل-مساحت ارایه می‌دهیم. در این روش، ما $\mathrm{Err_d}(p_ip_j)$ را برای تمام میانبرهای p_ip_j در زمان $O(n^2)$ محاسبه می‌کنیم. سپس ما از الگوریتم عمومی ایمایی و ایری (بخش ۲.۲ را ببینید) استفاده نموده و ساده‌سازی بهینه تحت معیار خطای تفاضل-مساحت را در زمان $O(n^2)$ پیدا می‌کنیم. روش‌های مختلفی برای محاسبه مساحت یک چندضلعی ساده وجود دارد [۴۲]. الگوریتم ما مبتنی بر روشی به نام رابطه‌ی قطبی[1] است. $\overrightarrow{p_ip_j}$ را یک ضلع جهت‌دار از

[1] Polar formula



شکل ۵.۴: $\mathrm{Ar}(A) = |\mathrm{Ar}(A_۱)| + |\mathrm{Ar}(A_۲)| + |\mathrm{Ar}(A_۳)| - |\mathrm{Ar}(A_۴)| - |\mathrm{Ar}(A_۵)|$

p_i به p_j در نظر بگیرید. برای هر ضلع $\overrightarrow{p_ip_j}$، $\mathrm{Ar}(\overrightarrow{p_ip_j})$ به صورت $x(p_i)y(p_j) - x(p_j)y(p_i)$ تعریف می‌شود که در آن $x(p)$ مولفه x از p و $y(p)$ مولفه y آن است. ثابت شده است که $|\mathrm{Ar}(\overrightarrow{p_ip_j})|$ دو برابر مساحت مثلث تشکیل شده توسط نقطه‌های p_i، p_j و $(۰, ۰)$ است [۴۲].

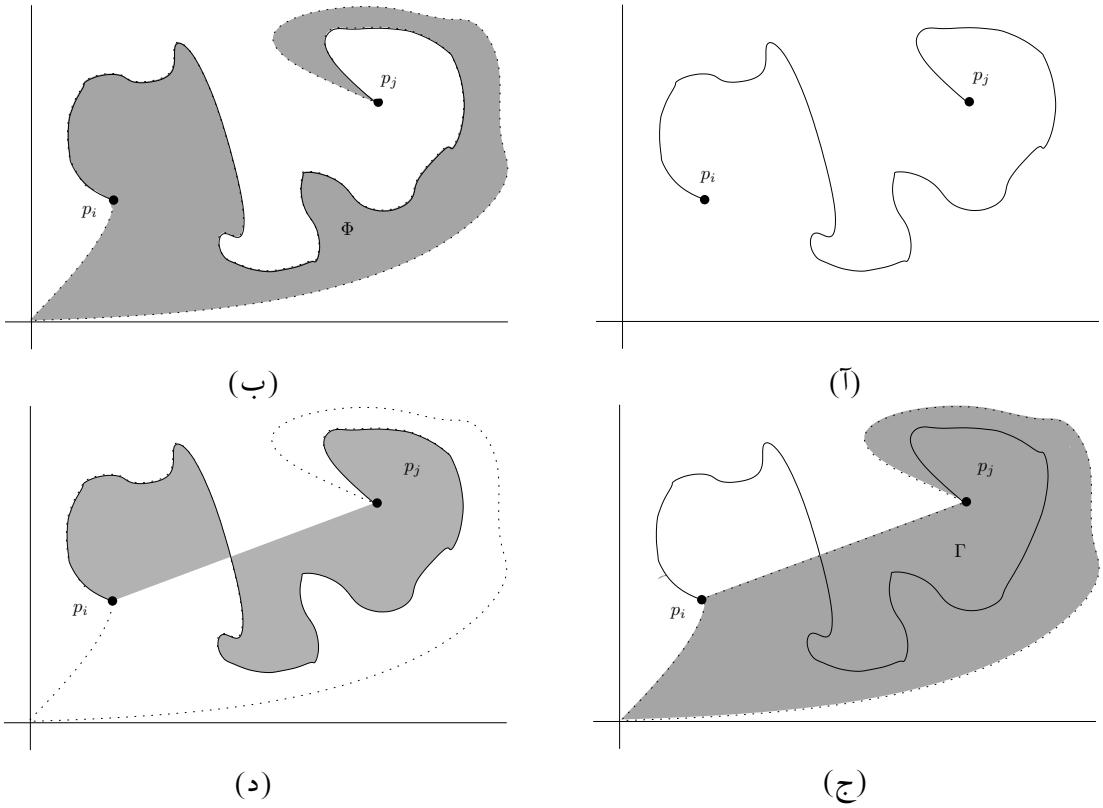
با استفاده از این موضوع، مساحت یک چندضلعی $Y = \{y_۱, ..., y_n\}$ با $e_i = y_iy_{i+۱}$، $۰ \leq i < n$ و $e_n = y_ny_۱$ را می‌توان با استفاده از رابطه زیر محاسبه نمود [۴۲]:

$$\mathrm{Ar}(Y) = |\frac{۱}{۲}\sum_{i=۰}^{n}\mathrm{Ar}(e_i)| \qquad (۱.۴)$$

نمونه‌ای از این محاسبه در شکل ۵.۴ آمده است . ما از این نتیجه برای محاسبه $\mathrm{Err_d}(p_ip_j)$ استفاده می‌کنیم. به عبارت دقیق‌تر، ما نشان می‌دهیم که اگر مسیر ورودی $\mathcal{P}$ ساده باشد، آنگاه $\mathrm{Ar}(Y = \mathcal{P}(i,j)) = \mathrm{Err_d}(p_ip_j)$ است و این بدان معنی است که رابطه (۱.۴) با معیار خطای تفاضل-مساحت که در بخش ۸.۲ تعریف شده سازگار است. در مرحله اول، با استفاده از تعدادی نقطه اضافی ما مبدا o را به p_i متصل می‌کنیم، به طوری که آن‌ها مسیر $\mathcal{P}(i,j)$ را قطع نکنند. $S = \{s_۰, s_۱, ..., p_k\}$ را مجموعه این نقطه‌ها قرار دهید. ما همین کار را برای اتصال p_j به o با استفاده از دنباله نقطه‌های $T = \{t_۰, t_۱, ..., t_l\}$ انجام می‌دهیم (شکل ۶.۴ب را ببینید). بدین ترتیب ما یک چندضلعی ساده $\mathcal{P}' = \{o, s_۰, s_۱, ..., s_k, p_i, p_{i+۱}, ..., p_j, t_۰, t_۱, ..., t_l\}$ را می‌سازیم که یک نقطه در مبدا دارد. Φ را برابر مساحت چندضلعی مشخص شده توسط $\mathcal{P}'$ قرار دهید. از رابطه ۱.۴ داریم:

$$\Phi = \mathrm{Ar}(\mathcal{P}') = \frac{۱}{۲}\sum_{i=۰}^{j-i+k+l+۲}\mathrm{Ar}(e_i') \qquad (۲.۴)$$

حال چندضلعی $\mathcal{P}'' = \{t_l, t_{l-۱}, ...t_۰, p_j, p_i, s_k, s_{k-۱}, ..., s_۰\}$ که توسط میانبر $\overrightarrow{p_jp_i}$ و نقطه‌های T و S



شکل ۶.۴: محاسبه $\mathrm{Err_d}(p_ip_j)$.

تشکیل یافته را در نظر بگیرید (شکل ۶.۴ج را ببینید). Γ را برابر مساحت چندضلعی مشخص شده توسط $\mathcal{P}''$ قرار دهید.

$$\Gamma = \mathrm{Ar}(\mathcal{P}'') = \frac{۱}{۲}\sum_{i=۰}^{l+k+۴} \mathrm{Ar}(e_i'') \qquad (۳.۴)$$

به آسانی می‌توان دید که:

$$\Gamma = \Phi - \mathrm{Ar}_l(p_ip_j) + \mathrm{Ar}_r(p_ip_j) \qquad (۴.۴)$$

بنابراین داریم:

$$\begin{aligned} |\Gamma - \Phi| &= |\mathrm{Ar}_r(p_ip_j) - \mathrm{Ar}_l(p_ip_j)| \\ &= \mathrm{Err_d}(p_ip_j) \end{aligned}$$

با استفاده از این نتیجه، ما یک روش ساده برای محاسبه کلیه $O(n^۲)$ مقدار $\mathrm{Err_d}(p_ip_j)$ در زمان $O(n^۲)$ ارایه می‌دهیم که $O(n)$ مرتبه نسبت به روش ابتدایی بهبود دارد. فرض کنید برای $۰ \le i < j \le n$، ما کلیه مقدارهای زیر را داریم:

$$S_{i,j} = \frac{۱}{۲}\sum_{k=i}^{j-۱} \mathrm{Ar}(p_kp_{k+۱}) \qquad (۵.۴)$$

ما نقطه آغازین i را ثابت در نظر گرفته و کلیه مقدارهای $S_{i,j}$ را برای کلیه jها، $j > i$ در زمان خطی محاسبه می‌کنیم. کل زمان لازم برای محاسبه $O(n^۲)$ مقدار مختلف $S_{i,j}$ برابر $O(n^۲)$ است. ولی با داشتن مقدارهای $S_{i,j}$، ما می‌توانیم $\mathrm{Err_d}(p_ip_j)$ را در زمان $O(n^۲)$ برای کلیه میانبرهای p_ip_j محاسبه کنیم. زیرا:

$$\mathrm{Err_d}(p_ip_j) \quad = \quad |S_{i,j} + \mathrm{Ar}(p_jp_i)|$$

بنابراین ما لم زیر را به دست می‌آوریم:

لم ۱.۳.۴. *محاسبه $\mathrm{Err_d}(p_ip_j)$ را برای کلیه $O(n^۲)$ میانبر از یک مسیر عمومی $\mathcal{P}$ را می‌توان در زمان $O(n^۲)$ به دست آورد.*

بنابراین ما قضیه زیر را خواهیم داشت

قضیه ۲.۳.۴. *الگوریتمی وجود دارد که با استفاده از آن می‌توان مسئله ساده‌سازی بهینه تحت معیار خطای تفاضل- مساحت را در زمان $O(n^۲)$ به دست آورد.*

۴.۴ جمع‌بندی

در این فصل ما مسئله ساده‌سازی مسیر را تحت معیارهای جمع- مساحت و تفاضل- مساحت مورد بررسی قرار دادیم. الگوریتم‌های قبلی در این زمینه یا دارای زمان اجرای بالا بودند یا آن که تنها بر روی مسیرهای x- یکنوا کار می‌کردند. به علت وجود تعریف‌های مختلف و غیریکپارچه برای معیار مساحت، ما ابتدا یک تعریف عمومی برای معیار مساحت و برای مسیرهای عمومی ارایه دادیم. این تعریف یکپارچه، با معیارهای خطای جمع- مساحت و تفاضل- مساحت سازگار است. برای تعریف ارایه شده و برای مسئله ساده‌سازی #- کمینه و تحت دو معیار نامبرده، برای شروع، ما یک الگوریتم ابتدایی با مرتبه زمانی $O(n^۳)$ ارایه نمودیم. سپس، ما یک الگوریتم تقریبی با مرتبه زمانی نزدیک به درجه دو برای مسئله ساده‌سازی یک مسیر عمومی دوبعدی تحت معیار جمع- مساحت ارایه نمودیم. همچنین، ما یک الگوریتم کارا برای مسئله ساده‌سازی مسیر تحت معیار تفاضل- مساحت ارایه نمودیم.

فصل ۵

الگوریتم‌های ساده‌سازی هم‌فضا

مسئله‌ی ساده‌سازی هم‌فضای مسیر یکی از مهمترین مسایل باز در زمینه ساده‌سازی هندسی بوده است. در این فصل ما یک چارچوب عمومی برای حل مسئله ساده‌سازی هم‌فضا و قویا- هم‌فضا ارایه می‌دهیم. این چارچوب مستقل از معیار خطا است و می‌تواند برای ساده‌سازی مسیر تحت هر معیار خطای دلخواهی مورد استفاده قرار گیرد. در ابتدا ما به مسئله ساده‌سازی قویا- هم‌فضا می‌پردازیم و الگوریتمی برای مسیرهای x- یکنوا ارایه می‌کنیم. سپس مسئله را برای مسیرهای عمومی بررسی نموده و الگوریتم و داده‌ساختاری را برای حل آن ارایه می‌دهیم. در ادامه به مسئله ساده‌سازی هم‌فضا پرداخته و الگوریتم بهینه‌ای برای مسیرهای عمومی ارایه می‌دهیم. در برخی کاربردها زمان عامل مهمتری نسبت به بهینه بودن اندازه پاسخ است. برای چنین موقعیت‌هایی ما یک الگوریتم مکاشفه‌ای برای ساده‌سازی هم‌فضای مسیرهای عمومی ارایه می‌دهیم. اگر چه این الگوریتم بهینه بودن اندازه پاسخ را تضمین نمی‌کند ولی زمان اجرای بسیار بهتری نسبت روش بهینه دارد. الگوریتم قویا- هم‌فضای ارایه شده برای مسیر x- یکنوا قبل از رفتن به فرصت مطالعاتی انجام شد. الگوریتم قویا- هم‌فضا برای مسیرهای عمومی در طی فرصت مطالعاتی و با مشارکت آقای دکتر آبام و آقای دکتر دلوران انجام شد. الگوریتم چند جمله‌ای برای مسیرهای هم‌فضا پس از بازگشت از فرصت مطالعاتی و با مشارکت آقای شایان احسانی انجام شد. الگوریتم‌های ارایه شده ما در کنفرانس‌ها و ژورنال زیر پذیرفته و یا چاپ شده‌اند. لازم به ذکر است که ژورنال حاصل از این کار پس از حدود دو سال و نیم داوری سرانجام در بهمن ماه امسال پذیرفته شد.

1. M.A. Abam, S.Daneshpajouh, L. Deleuran, M. Ghodsi, S. Ehsani, Computing Homotopic Line Simplification, Accepted in Elsevier journal of Computational Geometry: Theory & Applications, Jan. 2014

2. S. Daneshpajouh, M. Ghodsi, A Heuristic Homotopic Path Simplification Algorithm. ICCSA, Lecture Notes in Computer Science, 2011: 132–140

3. S. Daneshpajouh, M.A. Abam, L. Deleuran, M. Ghodsi, Computing Strongly Homotopic Line Simplification in the Plane. European Workshop on Computational Geometry (EuroCG), 185–188, 2011

فرض کنید $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ مجموعه نقطه‌های مسیر چندضلعی داده شده باشد. همچنین فرض کنید مسیر ساده است و خود را قطع نمی‌کند. یک مسیر چندضلعی $\mathcal{Q} = \{q_1 = p_1, q_2, \ldots, q_b = p_n\}$ با $b < n$، را یک ساده‌سازی از مسیر $\mathcal{P}$ می‌نامیم. در اینجا ما گونه محدود مسئله را بررسی می‌کنیم که در آن مجموعه نقطه‌های $\mathcal{Q}$ زیرمجموعه‌ای از مجموعه نقطه‌های $\mathcal{P}$ هستند. همچنین $S = \{s_1, s_2, \ldots, s_m\}$ را مجموعه نقطه‌های مانع در صفحه در نظر می‌گیریم، با این فرض که این نقطه‌ها و مسیر $\mathcal{P}$ با یکدیگر برخورد ندارند. هدف ما در اینجا یافتن یک ساده‌سازی هم‌فضا دارای کمینه نقطه‌های ممکن نسبت به مسیر اصلی تحت معیار دلخواه، برای گونه محدود و بیشینه خطای مسیر ($\max_{i=0}^{b-2} \mathrm{Err}_m(q_i q_{i+1})$) است.

۱.۵ الگوریتم‌های بهینه برای مسئله ساده‌سازی قویا- هم‌فضای مسیر

الگوریتم کلی ما به این صورت است که، ابتدا گراف G_ϵ بدون توجه به مانع‌ها ساخته می‌شود. سپس بررسی می‌شود که آیا میانبر $p_i p_j \in G_\epsilon$ نسبت به $\mathcal{P}(i,j)$ با در نظر گرفتن مانع‌ها، هم‌فضا است یا خیر. ما میانبرهای غیر- هم‌فضا را از G_ϵ حذف نموده و سپس کوتاه‌ترین مسیر از p_1 تا p_n را در G_ϵ پیدا می‌کنیم. مرحله آخر را می‌توان با یک جستجوی سطح- اول[1] در زمان $O(|G_\epsilon|$انجام داد. بدین ترتیب ساده‌سازی قویا- هم‌فضا با طول کمینه را پیدا می‌کنیم.

بنابراین، مسئله اصلی پیدا کردن میانبرهای هم‌فضا است (برای تعریف میانبر هم‌فضا به بخش ۴.۲ مراجعه کنید). این کار را می‌توان در زمان $O((n^3 + n^2 m)\log(nm))$ با استفاده از الگوریتم ارایه شده در [۲۵] و انجام آزمایش هم‌فضا بودن، انجام داد. در واقع کافی است به ازای هر میانبر $p_i p_j$، آزمایش هم‌فضا بودن را برای مسیر $p_1, ..., p_{i-1}, p_i, p_j, p_{j+1}, ..., p_n$ و مسیر اصلی $\mathcal{P}$ انجام داد. بنابراین پرسش اصلی این خواهد بود که آیا می‌توان تمام میانبرهای هم‌فضا را در زمان کارایی محاسبه نمود. ما به این پرسش پاسخ مثبت می‌دهیم. به عبارت روشن‌تر در ادامه، دو الگوریتم برای محاسبه میانبرهای قویا- هم‌فضا برای مسیرهای x- یکنوا و مسیرهای عمومی ارایه می‌دهیم. این الگوریتم‌ها به ترتیب در زمان $O(m\log(n+m) + n\log n\log(n+m) + k)$ و $O(n(m+n)\log(n+m))$ که در آن k تعداد میانبرهای هم‌فضا ، n طول مسیر و m تعداد مانع‌ها است، اجرا می‌شوند. نتیجه اصلی ما به صورت زیر است.

[1] BFS

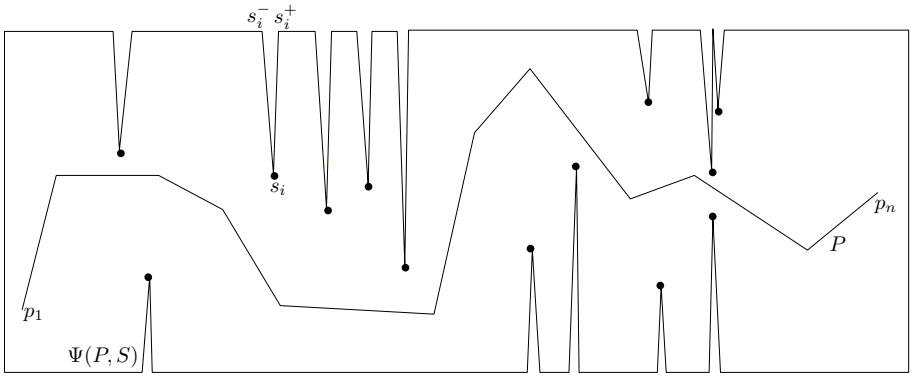
قضیه ۱.۱.۵. *فرض کنید $\mathcal{P}$ یک مسیر ساده به طول n در صفحه باشد به طوری که صفحه شامل m نقطه به عنوان مانع باشد. فرض کنید زمان موردنیاز برای محاسبه G_ϵ تحت تابع خطای F در غیاب مانع‌ها، $T_F(n)$ باشد. در این صورت (آ) اگر $\mathcal{P}$ یک مسیر x-یکنوا باشد، ساده‌سازی قویا-هم‌فضای بهینه را می‌توان در زمان $T_F(n) + O(m\log(m+n) + n\log n\log(n+m) + k)$، که در آن k تعداد میانبرهای هم‌فضا است، محاسبه نمود. (ب) اگر $\mathcal{P}$ یک مسیر عمومی باشد، ساده‌سازی قویا-هم‌فضای بهینه را می‌توان در زمان $T_F(n) + O(n(m+n)\log(n+m))$ محاسبه نمود.*

۱.۱.۵ الگوریتم بهینه برای محاسبه میانبرهای قویا-هم‌فضا برای مسیرهای x-یکنوا

ایده اصلی این الگوریتم این است که $\mathcal{P}$ را داخل یک چندضلعی ساده $\Psi(\mathcal{P}, S)$ به گونه‌ای محدود کنیم که اگر p_i بتواند p_j را در داخل $\Psi(\mathcal{P}, S)$ ببیند آنگاه p_ip_j یک میانبر هم‌فضا باشد. ما چندضلعی ساده $\Psi(\mathcal{P}, S)$ را به صورتی که در ادامه می‌آید تعریف می‌کنیم. ابتدا رابطه بالاتری مانع‌ها و مسیر $\mathcal{P}$ را مشخص می‌کنیم. مکان‌یابی هر مانع نسبت به مسیر $\mathcal{P}$ در زمان $O(\log n)$ قابل انجام است و در کل این کار را در زمان $O(m\log n)$ می‌توان انجام داد. سپس در زمان $O(n+m)$ یک مستطیل شامل مسیر و مانع‌ها را در داخل آن می‌سازیم. فرض کنید λ یک عدد به اندازه کافی کوچک و کمتر از نصف تفاضل مختصات x بین هر دو نقطه از مسیر و مانع‌ها باشد (λ را به راحتی می‌توان در زمان $O((n+m)\log(nm))$ محاسبه نمود). برای هر مانع s_i واقع در بالای مسیر $\mathcal{P}$، ما دو نقطه s_i^+ و s_i^- در ضلع بالایی مستطیل و با مقدار مختصات x به اندازه $x(s_i)+\lambda$ و $x(s_i)-\lambda$، که در آن $x(s_i)$ مقدار مختصات x از مانع s_i است، را تعریف می‌کنیم. ما s_i را به نقطه‌های s_i^+ و s_i^- متصل نموده و بخشی از مستطیل که s_i^+ را به s_i^- وصل کرده، حذف می‌کنیم. به طریق مشابه ما نقطه‌های s_i^+ و s_i^- و پاره‌خط‌های متناظر را برای مانع‌های واقع در زیر مسیر $\mathcal{P}$ به دست می‌آوریم. این‌ها با یکدیگر به ما چندضلعی ساده $\Psi(\mathcal{P}, S)$ را می‌دهد (شکل ۱.۵ را ببینید).

لم ۲.۱.۵. *یک میانبر p_ip_j یک میانبر هم‌فضا است، اگر و فقط اگر p_i بتواند p_j را در داخل $\Psi(\mathcal{P}, S)$ ببیند.*

اثبات. روشن است که با توجه به نحوه انتخاب λ، p_i در داخل $\Psi(\mathcal{P}, S)$ جای می‌گیرد. اگر p_i بتواند p_j را در درون $\Psi(\mathcal{P}, S)$ ببیند، آنگاه p_ip_j و $\mathcal{P}(i,j)$ یک حلقه را درون چندضلعی ساده $\Psi(\mathcal{P}, S)$ می‌سازند که این حلقه به راحتی می‌تواند تبدیل به یک نقطه شود و این بدان معنی است که p_ip_j و مسیر $\mathcal{P}(i,j)$ هم‌فضا هستند. حال فرض کنید که p_ip_j یک میانبر هم‌فضا است. حال طبق برهان خلف، فرض کنید p_i و p_j یکدیگر را در درون $\Psi(\mathcal{P}, S)$ نمی‌بینند. این بدان معنی است که p_ip_j باید هردو پاره خط $s_ks_k^-$ و $s_ks_k^+$ را برای برخی از



شکل ۱.۵: چندضلعی ساده $\Psi(\mathcal{P},S)$ که مسیر $\mathcal{P}$ را در بر گرفته است.

مقادیر k قطع کنند (توجه داشته باشید که هر دو نقطه p_i و p_j داخل $\Psi(\mathcal{P},S)$ هستند). این بدان معنی است که s_k بین میانبر p_ip_j و مسیر $\mathcal{P}(i,j)$ است که خلاف فرض ما برای هم‌فضا بودن p_ip_j است. □

لم فوق، مسئله ما را تبدیل به مسئله محاسبه گراف قابلیت دید n نقطه داخل یک چندضلعی ساده با اندازه $3m+4$ می‌کند. بن-موشه و همکارانش [۴۳] الگوریتمی برای محاسبه کلیه جفت نقطه‌هایی که یکدیگر را می‌بینند ارایه نموده‌اند. الگوریتم آن‌ها دارای پیچیدگی زمانی $O(m+n\log n\log(m+n)+k)$ و حافظه $O(m+n+k)$، که در آن k تعداد جفت نقطه‌هایی است که یکدیگر را می‌بینند، است. بنابراین ما نتیجه زیر را خواهیم داشت:

قضیه ۳.۱.۵. *فرض کنید $\mathcal{P}$ یک مسیر چندضلعی x-یکنوا باشد و S مجموعه‌ای شامل m نقطه به عنوان مانع روی صفحه باشد. کلیه میانبرهای هم‌فضا p_ip_j را می‌توان در زمان $O(m\log(m+n)+n\log n\log(n+m)+k)$ و با حافظه $O(n+m+k)$، که در آن k تعداد میانبرهای هم‌فضا است، محاسبه نمود.*

۲.۱.۵ الگوریتم بهینه برای محاسبه میانبرهای قویا-هم‌فضا برای مسیرهای عمومی

در این بخش به مسئله ساده‌سازی قویا-هم‌فضا برای مسیرهای عمومی می‌پردازیم. برای حل این مسئله ما یک داده‌ساختار درختی هندسی ارایه می‌دهیم. سپس الگوریتمی ارایه می‌دهیم که با استفاده از این داده‌ساختار، میانبرهای هم‌فضا را در زمان $O((m+n)\log(n+m))$ شناسایی می‌کند. در حل این مسئله از برخی ایده‌های [۲۵] استفاده شده است (برای آشنایی با آن به بخش ۱.۳.۳ مراجعه شود). فرض کنید $C_h=\{p_hp_i|1\le i\le n, h<i\le n\}$. در واقع C_h شامل همه میانبرهایی که با p_h آغاز می‌شوند است. در این صورت $\bigcup_{h=1}^{n}C_h$ شامل کلیه میانبرها خواهد بود. ما نشان می‌دهیم، برای h ثابت، چگونه می‌توان کلیه میانبرهای هم‌فضا در C_h را در زمان $O((m+n)\log(n+m))$ محاسبه نمود. این فرآیند برای کلیه $1\le h\le n$ قابل انجام است و بدین ترتیب کلیه میانبرهای هم‌فضا را می‌توان شناسایی نمود.

برای سادگی ارایه، در ادامه ما $h = ۱$ را ثابت در نظر می‌گیریم و به محاسبه میانبرهای هم‌فضا $C_۱$ در زمان $O((m+n)\log(n+m))$ می‌پردازیم. استراتژی کلی ما به صورت زیر است. در مرحله اول کلیه $\mathrm{CRP}(\mathcal{P}(۱,i))$ها (به اختصار $\mathrm{CRP}(i)$) را محاسبه نموده و آن‌ها را در صفحه $\mathcal{H}_۱$ نگهداری می‌کنیم. برای انجام این کار، ابتدا مسیر $\mathcal{P}$ را اصلاح می‌کنیم (برای آشنایی با مفهوم اصلاح کردن به بخش ۱.۳.۳ مراجعه نمایید). سپس با استقرا $\mathrm{CRP}(i)$ را برای همه iها محاسبه می‌کنیم. اگر $\mathrm{CRP}(i)$ دارای یک مانع دوری باشد، آنگاه نمی‌تواند با میانبر $p_۱p_i$ هم‌فضا باشد. زیرا CRP مربوط به میانبر $p_۱p_i$ دارای یک مانع دوری نیست. بنابراین ما $\mathrm{CRP}(i)$هایی را نگاه می‌داریم که x-یکنوا باشد. متاسفانه نگهداری کلیه $\mathrm{CRP}(i)$ها ممکن است نیاز به $O(n^۲)$ حافظه و زمان داشته باشد، که این به معنی داشتن یک الگوریتم غیر-کارا با پیچیدگی زمانی $O(n^۳)$ برای محاسبه کلیه میانبرهای هم‌فضا است. برای حل این مشکل ما از یک داده‌ساختار درختی هندسی با اندازه $O(n)$، که آن را $\mathcal{T}$ می‌نامیم، استفاده می‌کنیم. درخت $\mathcal{T}$ را در صفحه $\mathcal{H}_۱$ نگاه می‌داریم. در این صفحه هر $\mathrm{CRP}(i)$، یک مسیر از ریشه تا یک گره یا برگ در درخت $\mathcal{T}$ است (شکل ۲.۵ب). در مرحله بعد، در صفحه دیگری (صفحه $\mathcal{H}_۲$) ما همه میانبرهای $p_۱p_i$ $(۲ \le i \le n)$ که $\mathrm{CRP}(i)$ مربوط به آن‌ها x-یکنوا است را در حضور مانع‌ها اصلاح می‌کنیم. در نهایت به مرحله اصلی یعنی آزمون هم‌فضا بودن $\mathrm{CRP}(i)$ که در صفحه $\mathcal{H}_۱$ و در درخت $\mathcal{T}$ جای گرفته با میانبر اصلاح شده آن در صفحه $\mathcal{H}_۲$ می‌پردازیم. توجه داشته باشید که در شکل ۲.۵، $\mathrm{CRP}(۵)$ و $\mathrm{CRP}(۱۱)$، x-یکنوا نیستند، و در نتیجه در درخت $\mathcal{T}$ نگهداری نشده‌اند. ولی سایر میانبرها اصلاح شده‌اند و در شکل ۲.۵د آمده‌اند. باتوجه به آن که عمل اصلاح کردن مکان نقطه‌ها را در شکل ۲.۵ عوض می‌کند، برای هر نقطه p، ما از $\widehat{p}$ و $\widetilde{p}$ بجای p، درحالت اصلاح شده، استفاده می‌کنیم. توجه داشته باشید که عمل اصلاح کردن تنها مختصات y یک نقطه را عوض می‌کند و مختصات x آن بدون تغییر باقی می‌ماند. در ادامه کلیات الگوریتم ما را مشاهده می‌کنید.

۵-۱. الگوریتم محاسبه میانبرهای هم‌فضا

۱. مسیر $\mathcal{P}$ را اصلاح کن.

۲. یک ساختار پرسمان جابجایی-پاره‌خط [۲] بر روی مانع‌ها بساز.

۳. به ازای $h = ۱$ الی n، مرحله‌های ۴ تا ۷ را انجام بده.

۴. $\mathcal{H}_۱$، $\mathcal{H}_۲$ و $\mathcal{T}$ را برابر با تهی قرار بده.

۵. $\mathrm{CRP}(\mathcal{P}(h,i))$، $h < i \le n$، را محاسبه کن و آن‌ها را در درخت هندسی $\mathcal{T}$ در صفحه $\mathcal{H}_۱$ قرار بده.

[۲] Segment-dragging

۶. همه میانبرهای p_hp_i، $h < i \leq n$، را که $\mathrm{CRP}(\mathcal{P}(h,i))$ آن‌ها x-یکنوا است را در صفحه $\mathcal{H}_2$ اصلاح کن.

۷. آزمون هم‌فضا بودن هر $\mathrm{CRP}(\mathcal{P}(h,i))$، $h < i \leq n$، در $\mathcal{H}_1$ و میانبر اصلاح شده مربوط به آن در صفحه $\mathcal{H}_2$ را انجام بده.

۸. میانبرهای هم‌فضا را به عنوان نتیجه بازگردان.

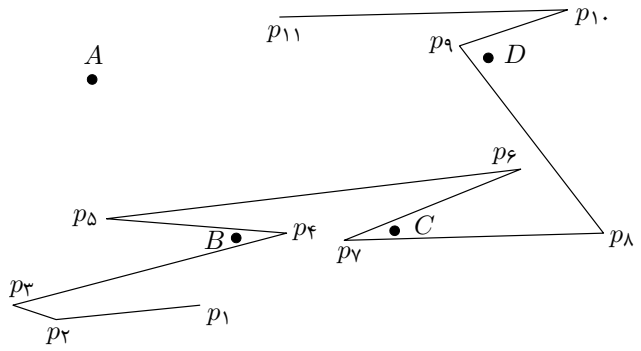
در ادامه به بیان جزئیات هر یک از مرحله‌های فوق می‌پردازیم.

اصلاح کردن $\mathcal{P}$

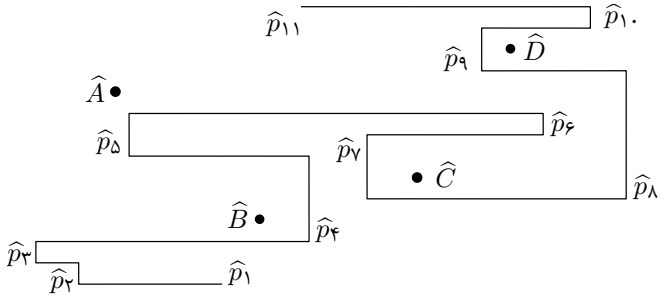
ابتدا ما مسیر $\mathcal{P}$ را اصلاح می‌کنیم. از آن جایی که $\mathcal{P}$ یک مسیر ساده است و خود را قطع نمی‌کند، ضلع‌های $\mathcal{P}$ و مانع‌های S یک رابطه بالایی-پایینی غیر حلقوی با یکدیگر می‌سازند. این رابطه را می‌توان در زمان $O((n+m)\log(n+m))$ محاسبه نمود [۴۴]. این رابطه بالایی-پایینی درواقع یک ترتیب جزئی را می‌سازد. این ترتیب جزئی را به آسانی می‌توان به یک ترتیب کامل تعمیم داد. $\mathrm{rank}_p(O)$ را رتبه شیء O (مانع یا لبه) در ترتیب کامل قرار دهید. با قرار دادن مختصات y هر عضوی از اشیاء O برابر با مقدار $\mathrm{rank}_p(O)$، مسیر $\mathcal{P}$ اصلاح می‌شود (برای مثال هر لبه توسط یک پاره‌خط افقی نشان داده می‌شود). ما دو پاره‌خط متعلق به دو لبه متوالی در $\mathcal{P}$ را توسط یک پاره‌خط عمودی بهم متصل می‌کنیم تا اتصال‌های شکل اصلی را حفظ کنیم (شکل ۲.۵ب را ببینید). توجه داشته باشید که برای هر پاره‌خط p_ip_{i+1}، یک پاره‌خط افقی در حالت اصلاح شده $\mathcal{P}$ است. ما سر پاره‌خط مربوط به p_{i+1} را، در حالت اصلاح شده، با $\widehat{p}_{i+1}$ نشان می‌دهیم.

پرسمان جابجایی-پاره‌خط

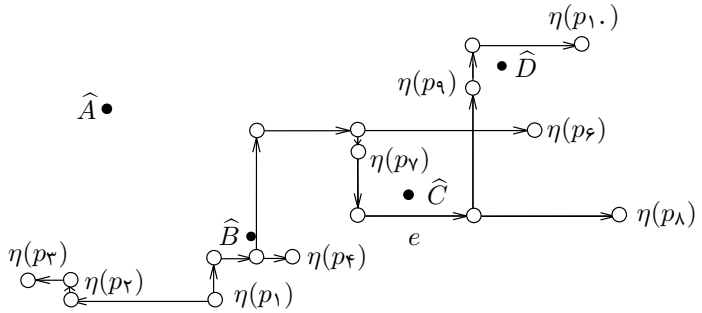
الگوریتم ما برای محاسبه $\mathrm{CRP}(i)$ مبتنی بر داده‌ساختار پرسمان جابجایی-پاره‌خط بر روی m نقطه به عنوان مانع است. این داده‌ساختار برای پیدا کردن اولین مانعی که با جابجایی پاره‌خط عمودی داده شده با آن برخورد می‌کند، مورد استفاده قرار می‌گیرد. از این پس ما چنین مانعی را نزدیک‌ترین مانع به پاره‌خط عمودی داده‌شده می‌نامیم. ما از داده‌ساختار شَزَل [۴۵] برای حل این مسئله استفاده می‌کنیم. این داده‌ساختار m مانع را در زمان $O(m\log m)$ پیش‌پردازش نموده و با استفاده از حافظه با اندازه $O(m)$ قادر است پرسمان‌های جابجایی-پاره‌خط را در زمان $O(\log m)$ پاسخ دهد.



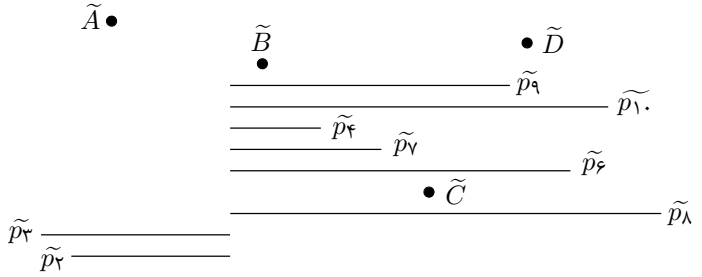
(آ) مسیر اصلی $\mathcal{P} = \{p_۱, \ldots, p_{۱۱}\}$.



(ب) مسیر اصلاح شده.



(ج) درخت $\mathcal{T}$در صفحه‌ی $\mathcal{H}_۱$. گره‌های $\mathcal{T}$ توسط دایره‌های توخالی نمایش یافته‌اند.



(د) میانبرهای اصلاح شده در صفحه‌ی $\mathcal{H}_۲$.

شکل ۲.۵: مرحله‌های کلی الگوریتم شناسایی میانبرهای قویا-هم‌فضا

مسیرهای اصلاح شده‌ی فشرده

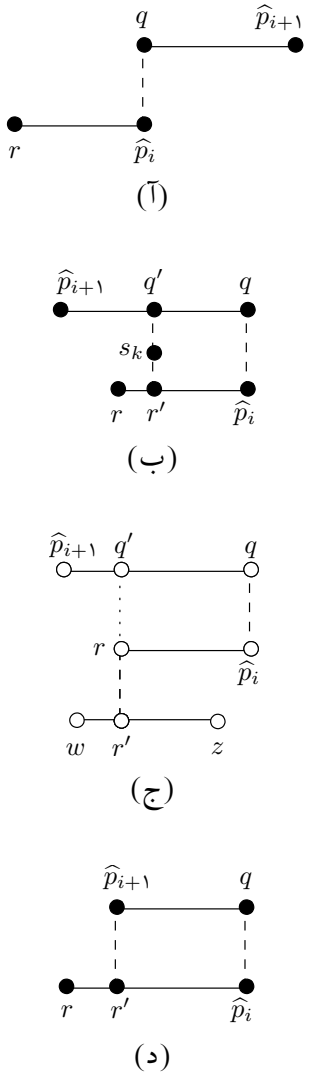
بعد از اصلاح کردن $\mathcal{P}$ و ساخت داده‌ساختار پرسمان- بازه- عمودی، ما آماده محاسبه کلیه $\mathrm{CRP}(i)$ها به صورت افزایشی هستیم. ما بر روی مسیر اصلاح شده‌ی $\mathcal{P}$ حرکت کرده و به ساخت درخت $\mathcal{T}$ می‌پردازیم. برای انجام این‌کار از دو پشته‌ی $\mathcal{S}_c$ و $\mathcal{S}_u$ استفاده می‌کنیم. در ابتدا، $\mathcal{S}_c$ برابر با اولین لبه از مسیر اصلاح شده $\mathcal{P}$ قرار داده شده و $\mathcal{S}_u$ برابر با تهی قرار داده می‌شود. همچنین درخت $\mathcal{T}$ را مقداردهی اولیه می‌کنیم: ما گره‌های جدید $\eta(p_1)$ و $\eta(p_2)$ و همچنین لبه جهت‌دار $(\eta(p_1), \eta(p_2))$ را به درخت خالی $\mathcal{T}$ اضافه نموده و $\eta(p_1)$ را ریشه $\mathcal{T}$ قرار می‌دهیم. در اینجا $\eta(p)$ بیانگر گره با برچسب p بوده و در محل نقطه p نشسته است.

به هنگام پردازش نقطه p_{i+1} از مسیر اصلاح شده‌ی $\mathcal{P}$، پشته $\mathcal{S}_c$، $\mathrm{CRP}(i)$ را به صورت دنباله‌ای از پاره‌خط‌های عمودی و افقی نگهداری می‌کند. همچنین پشته‌ی $\mathcal{S}_u$، دورهای U-شکلی را که تا رسیدن به $\mathrm{CRP}(i)$ دیده‌ایم را به صورت دنباله‌ای از پاره‌خط‌های عمودی نگهداری می‌کند. حال به محاسبه‌ی $\mathrm{CRP}(i+1)$ می‌پردازیم. پاره‌خط افقی از مسیر اصلاح شده $\mathcal{P}$ منتهی به نقطه p_{i+1} را در نظر بگیرید، نقطه انتهای دیگر این پاره‌خط را q بنامید (شکل ۳.۵ را ببینید). یک عنصر از بالای $\mathcal{S}_c$ ، که یک پاره‌خط منتهی به p_i است، را برمی‌داریم. سر دیگر آن را r می‌نامیم. اگر rp_i عمودی باشد، باید یک عنصر دیگر از پشته برداریم. حال فرض کنید، rp_i افقی است و p_i سمت راست r است. توجه کنید که p_i و q در مسیر اصلاح شده $\mathcal{P}$ توسط یک پاره خط عمودی متصل شده‌اند. ما چهارحالت را براساس سه گزاره‌ای که در ادامه می‌آیند تمییز می‌دهیم: (Q_1) $\widehat{p}_{i+1}$ در سمت راست $\widehat{p}_i$ واقع شده است؛ (Q_2) حداقل یک مانع بالای پاره‌خط $r\widehat{p}_i$ و پایین پاره‌خط $q\widehat{p}_{i+1}$ وجود دارد؛ (Q_3) طول $q\widehat{p}_{i+1}$ بزرگتر از طول $r\widehat{p}_i$ است (شکل ۳.۵ را ببینید که این چهار حالت را تفکیک کرده است). توجه داشته باشید که می‌توان درست یا نادرست بودن Q_2 را با استفاده از داده‌ساختار جابجایی- پاره‌خط بررسی نمود.

(آ) Q_1: پاره‌خط $r\widehat{p}_i$، $\widehat{p}_iq$ و $q\widehat{p}_{i+1}$ به بالای پشته $\mathcal{S}_c$ اضافه می‌شوند. به علاوه، اگر $\mathcal{S}_u$ خالی باشد، $\mathcal{T}$ را بروزرسانی می‌کنیم: ما گره‌های جدید $\eta(p_{i+1})$ و $\eta(q)$ و همچنین لبه‌های جدید $(\eta(p_i), \eta(q))$ و $(\eta(q), \eta(p_{i+1}))$ را به $\mathcal{T}$ اضافه می‌کنیم.

(ب) نقیض(Q_1)، فقط Q_2: $r'q'$ را پاره‌خط عمودی که به s_k از سمت راست تماس دارد، که در آن s_k سمت راست‌ترین مانع در سمت چپ $\widehat{p}_iq$ است، قرار دهید. ما rr'، $r'q'$ و $q'\widehat{p}_{i+1}$ را در $\mathcal{S}_c$ قرار می‌دهیم. درواقع ما $r'\widehat{p}_i$، $\widehat{p}_iq$ و qq' را حذف می‌کنیم. همچنین $r'q'$ را به $\mathcal{S}_u$ به عنوان یک دور U-شکل جدید اضافه می‌کنیم. از آن جایی که $\mathcal{S}_u$ خالی نیست، درخت $\mathcal{T}$ را بروزرسانی نمی‌کنیم.

(ج) نقیض(Q_1)، نقیض(Q_2)، فقط Q_3: rr' را پاره‌خط عمودی که از r آویزان شده و می‌تواند با خواندن



شکل ۳.۵: حالات مختلف با توجه به موقعیت $r\widehat{p}_j$ و $q\widehat{p}_{i+1}$ و مانع‌ها وقتی $\widehat{p}_i$ در سمت راست r است.

دوباره یک عنصر از بالای پشته $\mathcal{S}_c$ به دست آید، قرار دهید. ما $r'r$ و rq' را چسبانده و $r'q'$ و همچنین $q'\widehat{p}_{i+1}$ را به $\mathcal{S}_c$ اضافه می‌کنیم. به علاوه اگر بالای $\mathcal{S}_u$، $r'r$ باشد، ما $r'r$ را از $\mathcal{S}_u$ می‌خوانیم. اگر $\mathcal{S}_u$ خالی باشد، درخت $\mathcal{T}$ را بروزرسانی می‌کنیم: گره جدید $\eta(r')$ (اگر وجود نداشته باشد)، $\eta(q')$ و $\eta(p_{i+1})$ و همچنین لبه جهت‌دار جدید $(\eta(r'),\eta(q'))$ را به درخت $\mathcal{T}$ اضافه می‌کنیم. اگر $\eta(r')$ قبلا وجود نداشته باشد، باید لبه $(\eta(z),\eta(w))$ از درخت $\mathcal{T}$ که جاسازی آن در صفحه (برای مثال پاره‌خط zw) حاوی نقطه r' است را به دو لبه جهت دار $(\eta(z),\eta(r'))$ و $(\eta(r'),\eta(w))$ تقسیم کنیم. اگر $r'q'$ باز هم می‌توانست به سمت چپ حرکت کند (این موضوع را می‌توان در زمان ثابت با در نظر گرفتن سه پاره‌خط آخر از $\mathcal{S}_c$ بررسی نمود)، آنگاه $q'\widehat{p}_{i+1}$ را از بالای پشته برداشته و آن را به عنوان یک پاره‌خط افقی جدید برای اضافه شدن، در نظر گرفته و مرحله (ج) را تکرار می‌کنیم.

(د) نقیض(Q_1)، نقیض(Q_2)، و نقیض(Q_3) : rr' و $r'\widehat{p}_{i+1}$ را به $\mathcal{S}_c$ اضافه می‌کنیم. به علاوه اگر $\mathcal{S}_u$ خالی باشد، درخت $\mathcal{T}$ را بروزرسانی می‌کنیم. ما گره‌های جدید، $\eta(r')$ و $\eta(p_{i+1})$ را اضافه نموده و لبه $(\eta(p_i),\eta(r))$ را حذف نموده و لبه‌های جدید $(\eta(r),\eta(r'))$، $(\eta(r'),\eta(p_i))$ و $(\eta(r'),\eta(p_{i+1}))$ را اضافه می‌کنیم.

سایر حالت‌ها یا با حالت‌های فوق متقارن هستند یا با کمی تغییر می‌توان آن‌ها را محاسبه نمود. از آن جایی که در هر تکرار از n تکرار، یک تعداد ثابتی از لبه‌ها به درخت $\mathcal{T}$ اضافه شده‌اند، اندازه $\mathcal{T}$ برابر با $O(n)$ خواهد بود. توجه داشته باشید که در برخی تکرارها، مرحله (ج) ممکن است به صورت بازگشتی چندین بار اجرا شود، اما تنها برای آخرین تکرار آن، ممکن است لازم باشد درخت $\mathcal{T}$ را بروزرسانی کنیم و از طرف دیگر تعداد چنین بازگشتی‌هایی در کل برابر با $O(n)$ است. علت این است که هر بازگشتی می‌تواند مربوط به یک پاره‌خط افقی از مسیر باشد، و هر پاره‌خط افقی می‌تواند حداکثر دوبار ملاقات شود. بنابراین لم زیر را خواهیم داشت.

لم ۴.۱.۵. *کلیه $CRP(i)$های x-یکنوا را می‌توان در زمان $O(n\log m)$ در یک درخت $\mathcal{T}$ با اندازه $O(n)$ جاسازی نمود به طوری که در آن هر $CRP(i)$ x-یکنوا متعلق به یک مسیر از ریشه به یک گره یا برگ در درخت $\mathcal{T}$ باشد.*

اصلاح کردن میانبرها

I را مجموعه شاخص‌های i به گونه‌ای که $\mathrm{CRP}(i)$، x-یکنوا باشد، قرار دهید. به این ترتیب، I شامل باقی مانده میانبرهایی، پس از محاسبه کلیه $\mathrm{CRP}(i)$ها، خواهد بود که کاندید هم‌فضا بودن هستند. کلیه میانبرهای p_1p_i، $i\in I$، و m مانع در صفحه را در نظر بگیرید. آن‌ها یک رابطه بالایی-پایینی دارند و یک ترتیب جزئی را تشکیل می‌دهند. به راحتی می‌توان این رابطه را به یک ترتیب کامل تبدیل نمود. $\mathrm{rank}_s(O)$ را رتبه شیء O (مانع یا میانبر) در ترتیب کامل قرار دهید. ما مختصات y از هر شیء O را برابر با $\mathrm{rank}_s(O)$ قرار می‌دهیم. بدین‌ترتیب کلیه میانبرها اصلاح می‌شوند. ما میانبر اصلاح شده p_1p_i را با $\widetilde{p}_1\widetilde{p}_i$ نشان می‌دهیم (شکل ۲.۵د را ببینید). توجه داشته باشید که برای شلوغ نشدن تصویر، $\widetilde{p}_1$ در این شکل نوشته نشده است. همچنین در این شکل هیچ رابطه‌ای بین $\mathrm{rank}_p(s_i)$ و $\mathrm{rank}_s(s_i)$ برای مانع s_i وجود ندارد. زیرا این دو رتبه متعلق به دو «ترتیب کامل» متفاوت می‌باشند.

تست هم‌فضایی برای درخت $\mathcal{T}$ و میانبرها اصلاح شده

تا این جا دو صفحه داریم: (آ) صفحه $\mathcal{H}_1$ که شامل مانع‌ها و درخت $\mathcal{T}$ است و (ب) صفحه $\mathcal{H}_2$ که شامل مانع‌ها و میانبرهای اصلاح شده مربوط به $\mathrm{CRP}(i)$های x-یکنوا که در درخت $\mathcal{T}$ جاسازی شده‌اند، است.

درخت $\mathcal{T}$ دارای $O(n)$ لبه است و تمام لبه‌های آن در $\mathcal{H}_1$ جای گرفته‌اند. توجه داشته باشید که $\mathcal{T}$ ممکن است خودش را قطع کند، و در نتیجه یک ترتیب جزئی بین مسیرهای جاسازی شده در $\mathcal{T}$ لزوما وجود ندارد. مختصات y از همه میانبرهای افقی و مانع‌ها در $\mathcal{H}_1$ از rank_p می‌آید، حال آن که در $\mathcal{H}_2$ از rank_s می‌آید. همچنین یادآوری می‌شود که I مجموعه شاخص‌های i به نحوی که $\text{CRP}(i)$ x-یکنوا باشد است.

برای لبه افقی e از $\mathcal{T}$، above(e) را مجموعه مانع‌های s_j قرار دهید به طوری که (آ) $\text{rank}_p(s_j) > \text{rank}_p(e)$ و (ب) مختصات x مانع s_j، بین مختصات x دونقطهِ دو سر e قرار گیرد. مشابه حالت فوق below(e) را تعریف می‌کنیم به طوری که شامل کلیه مانع‌های پایین ضلع e باشد. با توجه به آن که هر مانع در بالا (یا پایین) $\text{CRP}(i)$ تنها بالای (یا پایین) دقیقا یک لبه از لبه‌های $\text{CRP}(i)$ است، آزمون هم‌فضا بودن p_1p_i و $\text{CRP}(i)$ را می‌توان به تعدادی آزمون هم‌فضا بودن تقسیم نمود به طوری که لبه‌هایی که در $\text{CRP}(i)$ هستند را شامل شود. برای هر لبه e که در $\text{CRP}(i)$ ظاهر می‌شود، هر مانعی در above(e) باید بالای $\widetilde{p}_1\widetilde{p}_i$ باشد و هر مانعی در below(e) باید پایین $\widetilde{p}_1\widetilde{p}_i$ باشد؛ در غیر این صورت، p_1p_i و $\text{CRP}(i)$ نمی‌توانند هم‌فضا باشند. بنابراین، برای هر لبه e و هر مانع s_j، شرط زیر را خواهیم داشت.

شرط ۵.۱.۵. $(s_j \in \text{above}(e) \wedge \text{rank}_s(p_1p_i) < \text{rank}_s(s_j)) \bigvee (s_j \in \text{below}(e) \wedge \text{rank}_s(p_1p_i) > \text{rank}_s(s_j))$.

لم زیر حالتی را که میانبر اصلاح شده $\widetilde{p}_1\widetilde{p}_i$ و $\text{CRP}(i)$ هم‌فضا هستند را بیان می‌کند:

لم ۶.۱.۵. *برای هر $i \in I$، میانبر اصلاح شده p_1p_i و $CRP(i)$ هم‌فضا هستند اگر و فقط اگر برای هر لبه افقی e از $\mathcal{T}$ که در $CRP(i)$ ظاهر می‌شود، و هر مانع $s_j \in above(e)$ ($s_j \in below(e)$) شرط ۵.۱.۵ ارضا شود.*

این لم این موضوع را بیان می‌کند که هر لبه e و هر شیء $s_j \in \text{above}(e)$ ($s_j \in \text{below}(e)$) که شرط ۵.۱.۵ را برآورده نسازد، میانبر p_1p_i را از هم‌فضا بودن خارج می‌کند و اگر چنین جفت e و s_j وجود نداشته باشد، میانبر p_1p_i قطعا هم‌فضا خواهد بود. با این حال آزمون شرط ۵.۱.۵ برای هر لبه e و هر $s_j \in \text{above}$ ($s_j \in \text{below}$) در کل پر هزینه است. لم زیر نشان می‌دهد که در میان تمام مانع‌های بالا و پایین e تنها دو مانع با e لازم است آزمایش شوند تا شرط ۵.۱.۵ برآورده شود.

لم ۷.۱.۵. *برای بررسی شرط ۵.۱.۵ تنها کافی است به ازای هر لبه افقی e از $\mathcal{T}$، آزمون را با مانع s_j که دارای کمترین (بیشترین) rank_s در $above(e)$ ($below(e)$) است انجام دهیم.*

اثبات. ما اثبات را برای above(e) انجام می‌دهیم. اثبات below(e) با استدلال مشابه قابل انجام است. O_c را شیء با رتبه $\min(\text{above}(e))$ قرار دهید. روشن است که تمام میانبرهایی که دارای rank_s بزرگتر از

$\text{rank}_s(O_c)$ هستند، بالای O_c قرار می‌گیرند. بنابراین، هیچ یک از چنین میانبرهایی نمی‌توانند با زیر مسیرشان هم‌فضا باشند. همچنین، تمام میانبرها با rank_s کوچکتر از $\text{rank}_s(O_c)$ در زیر O_c جای می‌گیرند. با توجه به آن که هیچ مانع دیگری زیر O_c قرار نمی‌گیرد، هر یک از این میانبرها می‌تواند با زیر مسیرش هم‌فضا باشد.

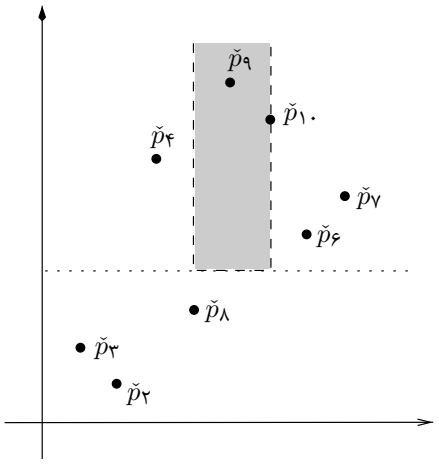
حال از برهان خلف استفاده می‌کنیم. فرض کنید که یک مانع O_z وجود دارد که باعث می‌شود برخی میانبرها غیر-هم‌فضا شوند. اگر این میانبرها در بالای O_c جای بگیرند، آنگاه آن‌ها به طور واضح میانبرهای غیر-هم‌فضا هستند و نیازی به آزمایش سایر مانع‌ها نداریم. بنابراین، فرض کنید که برخی از این میانبرها زیر O_c قرار می‌گیرند. در این حالت، با توجه به آن که O_z زیر چنین میانبرهایی است، داریم: $\text{rank}_s(O_z) < rank_s(O_c)$. این با فرض اولیه ما که O_c نزدیک‌ترین مانع به e است، تناقض دارد. □

با دانستن لم فوق، سوال اصلی این خواهد بود که آزمون ۵.۱.۵ برای حذف لبه‌های افقی e از $\mathcal{T}$ را با چه هزینه زمانی می‌توانیم انجام دهیم. برای حذف میانبرهای غیرهم‌فضا، همان طور که می‌دانیم above(e) یا below(e) ممکن است تعداد زیادی مانع وجود داشته باشد و $I(e)$ ممکن است شامل چندین شاخص باشد ($I(e)$ شامل مجموعه شاخص‌های $i \in I$ به طوری‌که e در $\text{CRP}(i)$ باشد است). در ادامه توجه‌مان را بر روی above(e) قرار می‌دهیم (below(e) به همین طریق قابل بررسی خواهد بود).

یک راه آسان برای محاسبه min(above(e)) برای هر لبه افقی e از $\mathcal{T}$ این است که یک داده‌ساختار پرسمان بازه-عمودی $\mathcal{T}_{\text{obs}}$ بر روی مانع‌ها در $\mathcal{H}_1$ بسازیم. چنین درختی، یک درخت دو سطحی خواهد بود. برای یک گره ν در سطح دوم از $\mathcal{T}_{\text{obs}}$، اطلاعات اضافی تحت عنوان کمینه rank_s از بین رتبه مانع‌های واقع در زیر-درخت با ریشه ν، نگهداری می‌کنیم. بنابراین min(above(e)) را می‌توان در زمان $\log^2 m$ به دست آورد. با این حال با توجه به آن که همه لبه‌ها را از قبل داریم، این کار را می‌توان در زمان سریع‌تری انجام داد. می‌توانیم لبه‌های افقی و مانع‌ها را از بالا به پایین جاروب کنیم و یک درخت جستجوی دودویی بر روی مانع‌های جاروب شده براساس مختصات x آن‌ها، نگهداری کنیم. وقتی خط جاروب به یک لبه e می‌رسد، above(e) را می‌توان به عنوان اجتماع $O(\log m)$ زیر درخت در درخت جستجوی دودویی دید و در نتیجه min(above(e)) را می‌توان در زمان $O(\log m)$ محاسبه نمود.

بررسی شرط ۵.۱.۵

برای این که به صورت دسته جمعی شرط ۵.۱.۵ را بررسی کنیم، یک ترتیب جدید از اشیاء در I به قسمی که عناصر واقع در $I(e)$ به صورت متوالی قرار گیرند، تعریف می‌کنیم. این ترتیب با پیمایش میان-ترتیب روی $\mathcal{T}$ به دست می‌آید. توجه داشته باشید که به ازای هر $i \in I$ یک گره در $\mathcal{T}$ با برچسب $\eta(p)_i$ وجود دارد. $\sigma(i)$ را رتبه $i \in I$ در این ترتیب قرار دهید. به عنوان مثال، $\langle ۳, ۲, ۴, ۸, ۹, ۱۰, ۶, ۷ \rangle$ ترتیب جدید شاخص‌ها



شکل ۴.۵: نقطه‌های جدید $(\sigma(i), \mathrm{rank}_s(p_۱p_i))$ در صفحه $\mathcal{H}$ برای شاخص‌های آمده در شکل ۲.۵ج. ناحیه خاکستری یک بازه ۳-وجهی مربوط به $\mathcal{I}(e) = \{۸, ۹, ۱۰\}$ است. با توجه به آن که نقطه‌های $\check{p}_۹$ و $\check{p}_{۱۰}$ در ناحیه خاکستری واقع شده‌اند، آن‌ها شرط ۵.۱.۵ را نقض می‌کنند. بنابراین، $\check{p}_۹$ و $\check{p}_{۱۰}$ نمی‌توانند میانبرهای هم‌فضا باشند. توجه داشته باشید که در این ترتیب جدید، پاره‌خط مربوط به e در $\mathcal{H}$ عبارت است از $[۴, ۶]$. همچنین $\min(\mathrm{above}(e))$ برابر ۴، که عبارت است از $\mathrm{rank}_s(\widetilde{C})$.

در $\mathcal{T}$ مربوط به شکل ۲.۵ج است. برای لبه e در این شکل، $\mathcal{I}(e)$ عبارت است از $\{۸, ۹, ۱۰\}$ که در آن $\mathcal{I}(e)$، در ترتیب جدید، متوالی هستند (در بخش‌های جدا از هم قرار نگرفته‌اند). این خصوصیت باعث کاهش زمان بررسی شرط ۵.۱.۵ برای لبه e می‌شود. برای هر $i \in I$، نقطه $\check{p}_i = (\sigma(i), \mathrm{rank}_s(p_۱, p_i))$ را در یک صفحه جدید $\mathcal{H}$ قرار می‌دهیم (شکل ۴.۵ را ببینید). هر لبه e و $\min(\mathrm{above}(e))$ مربوط به آن یک بازه عمودی ۳-وجهی در $\mathcal{H}$ تعریف می‌کند. توجه داشته باشید که پاره‌خط مربوط به e می‌تواند در پیمایش میان-ترتیب $\mathcal{T}$ محاسبه شود. همچنین $\min(\mathrm{above}(e))$ یک نیم-خط موازی محور xها تعریف می‌کند. پاره‌خط مربوط به e و نیم‌خط مربوط به $\min(\mathrm{above}(e))$، به اتفاق، یک بازه جستجوی ۳-وجهی تعریف می‌کنند که همه شاخص‌هایی را که شرط ۵.۱.۵ را نقض می‌کنند را مشخص می‌کند. شکل ۴.۵ یک بازه ۳-وجهی برای لبه e مشخص شده در شکل ۲.۵ج را نشان می‌دهد. در نتیجه هر $i \in \mathcal{I}$ که نقطه متناظرش در این بازه قرار می‌گیرد، شرط ۵.۱.۵ را نقض می‌کند و در نتیجه باید از $\mathcal{I}$ حذف شود. از آن جایی که این بازه‌های ۳-وجهی جستجو از قبل موجود هستند، می‌توانیم نقطه‌ها و بازه‌ها را در $\mathcal{H}$ از بالا به پایین جاروب کنیم و یک درخت جستجوی دودویی روی نقطه‌ها با توجه به مختصات x آن‌ها (مشابه آنچه در پاراگراف قبلی گفته شد) تعریف کنیم. به هنگام پردازش بازه سه وجهی، ما $O(\log n)$ زیردرخت را از درخت دودویی حذف می‌کنیم. بنابراین عمل جاروب کردن در کل $O(n \log n)$ زمان نیاز خواهد داشت. سپس این پردازش برای $\mathrm{below}(e)$ برای لبه‌های افقی e از $\mathcal{T}$ به روش مشابه، انجام می‌شود. هر میانبر باقی مانده یک میانبر هم‌فضا است. با کنارهم گذاشتن همه این‌ها باهم، نتیجه زیر را خواهیم داشت.

قضیه ۸.۱.۵. *$\mathcal{P} = \{p_۱, \ldots, p_n\}$ را یک مسیر چندضلعی ساده در نظر گرفته و S را مجموعه نقطه‌ها شامل*

m مانع در صفحه در نظر بگیرید. تمام میانبرهای هم‌فضا p_ip_j را می‌توان در زمان $O(n(m+n)\log(m+n))$ و با حافظه $O(n+m+k)$، که در آن k تعداد میانبرهای هم‌فضا است، محاسبه نمود.

۲.۵ الگوریتم بهینه برای ساده‌سازی هم‌فضای مسیرهای عمومی

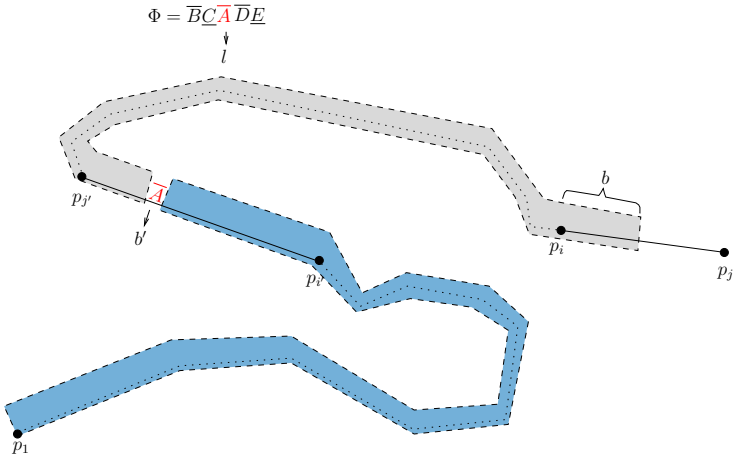
فرض کنید $\mathcal{P} = \{p_1, \ldots, p_n\}$ یک مسیر چند ضلعی به همراه m نقطه (مانع) در یک صفحه است (مسیر می‌تواند با خودش برخورد داشته باشد). در این بخش، الگوریتمی برای پیداکردن یک ساده‌سازی بهینه برای مسیرهای عمومی را ارایه می‌دهیم. ایده کلی ما این است که مسیر $\mathcal{P}$ و هر ساده‌سازی از آن را می‌توان به صورت رشته‌ها، یا دقیق‌تر با دنباله کانونی آن‌ها، مدل کرد. با چنین نگاهی، هدف یافتن یک زیردنباله از میانبرهای $\{q_1 = p_1, q_2, \ldots, q_{n-1}, q_n = p_n\}$ $\mathcal{Q} =$ است به طوری که تعداد میانبرهای آن کمینه و $\mathrm{CS}(\mathcal{Q}) = \mathrm{CS}(\mathcal{P})$ باشد. ما از تکنیک برنامه‌ریزی پویا برای پیدا کردن زیردنباله کمینه از میانبرها استفاده می‌کنیم. برای سادگی، از این پس به جای عبارت *زیردنباله از میانبرها*، عبارت *زیردنباله* را به کار می‌بریم.

ابتدا G_ϵ را تحت تابع فاصله‌ی F محاسبه می‌کنیم. سپس به ازای هر لبه p_ip_j در G_ϵ مقدار $\mathrm{STR}(p_ip_j)$ آن را حساب می‌کنیم. برای محاسبه‌ی STR از روشی مشابه محاسبه‌ی CS یک مسیر استفاده می‌کنیم. تنها تفاوت این است که پس از به دست آمدن رشته، عمل حذف نمادهای تکراری را روی آن انجام نمی‌دهیم. درواقع یک میانبر نمی‌تواند دارای نمادها تکراری باشد و در نتیجه $\mathrm{CS}(p_ip_j) = \mathrm{STR}(p_ip_j)$ خواهد بود. همچنین از $\mathrm{STR}(b, p_ip_j)$ و $\mathrm{STR}(p_ip_j, b)$ به ترتیب برای نشان دادن b نماد ابتدا و b نماد انتهای $\mathrm{STR}(p_ip_j)$ استفاده می‌کنیم. همچنین $\mathrm{CS}(\mathcal{P})$ را هم به دست می‌آوریم. ما k-امین نماد از $\mathrm{CS}(\alpha)$ را با $\mathrm{CS}(\alpha)[k]$ نشان می‌دهیم. به طریق مشابه k-امین نماد از $\mathrm{STR}(\alpha)$ را با $\mathrm{STR}(\alpha)[k]$ نشان می‌دهیم. وقتی تمام $\mathrm{STR}(p_ip_j)$ها برای همه‌ی i و jهای ممکن، در دسترس باشند، CS مربوط به زیردنباله‌ی $\mathcal{Q}$ را می‌توان به آسانی با اتصال STR میانبرهای واقع در $\mathcal{Q}$ و سپس حذف عنصرهای تکراری همسایه به دست آورد. به بیان دقیق‌تر داریم:

$$\mathrm{CS}(\mathcal{Q}(n)) = \begin{cases} \mathrm{RM}(\mathrm{CS}(\mathcal{Q}(n-1) \oplus \mathrm{STR}(q_{n-1}q_n)) & \text{اگر } n > 1 \\ \mathrm{STR}(q_1q_2) & \text{اگر } n = 2 \end{cases}$$

که در آن $\mathcal{Q}(n) = \{q_1, q_2, \ldots, q_n\}$ و عمل‌گر $\oplus$ عمل اتصال بین دو رشته و RM عمل حذف نمادهای تکراری همسایه را انجام می‌دهد (به عنوان مثال $\mathrm{RM}(\overline{C}\underline{A}\overline{AB}B\underline{AC})$ برابر است با $\overline{C}\underline{C}$).

الگوریتم پویای پیشنهاد شده، براساس این مشاهده است که، اگر $\mathrm{CS}(\mathcal{Q}) = \mathrm{CS}(\mathcal{P})$ باشد، آنگاه باید i و jی وجود داشته باشد که هر پیشوند از $\mathrm{CS}(\mathcal{P})$ را بتوان با اتصال STR مربوط به $i-1$ میانبر ابتدای $\mathcal{Q}$ و j نماد ابتدای STR مربوط به i-امین میانبر $\mathcal{Q}$، و حذف عنصرهای تکراری، به دست آورد. درواقع، آخرین نماد از پیشوند $\mathrm{CS}(\mathcal{P})$ باید با یک نماد از نمادهای $\mathrm{CS}(\mathcal{Q})$ همخوانی داشته باشند. بنابراین این نماد باید در یکی



شکل ۵.۵: فرض کنید $\Phi = \mathrm{CS}(\mathcal{P})$ باشد و $\mathcal{Q}$ را زیر ساده‌سازی که $\mathrm{OptHS}[i,j,b,l]$ را مشخص می‌کند در نظر بگیرید. در این صورت باید میانبر $p_{i'}p_{j'}$ی در $\mathcal{Q}$ باشد که برای برخی مقادیر b'، $\mathrm{STR}(p_{i'}p_{j'})[b'] = \mathrm{CS}(\mathcal{P})[l]$ بوده و $\mathrm{CS}(\mathrm{seq}(i',j',i,j,|\mathrm{STR}(p_{i'}p_{j'})|-b',b))$ تهی باشد.

از STRهای میانبرهای $\mathcal{Q}$ (i-امین میانبر برای برخی iها) وجود داشته باشد (زیرا $\mathrm{CS}(\mathcal{Q})$ از پیوستن STR میانبرهایش به دست می‌آید). اگر موقعیت این نماد در STR مربوط به i-امین میانبر $\mathcal{Q}$ برابر j باشد، آنگاه پیشوند $\mathrm{CS}(\mathcal{P})$ برابر است با :

$$\mathrm{RM}(\mathrm{CS}(Q(i-۱)) \oplus \mathrm{STR}(j, q_iq_{i+۱}))$$

به عنوان مثال فرض کنید $\mathrm{CS}(\mathcal{P}) = \overline{B}\underline{C}\overline{A}\overline{D}\underline{E}$ باشد. همچنین $\delta = \overline{B}\underline{C}\overline{A}$ را پیشوندی از $\mathrm{CS}(\mathcal{P})$ در نظر بگیرید. در این صورت باید میانبر $q_iq_{i+۱} \in \mathcal{Q}$ وجود داشته باشد که مثلا j-امین نماد آن ($\mathrm{STR}(j, q_iq_{i+۱})$) برابر با آخرین نماد پیشوند δ یعنی $\overline{A}$ باشد و همچنین $\mathrm{RM}(\mathrm{CS}(Q(i-۱)) \oplus \mathrm{STR}(j, q_iq_{i+۱}))$ برابر با $\overline{B}\underline{C}$ باشد.

همچنین $\mathrm{seq}(\mathcal{Q}, b)$ را به صورت زیر تعریف می‌کنیم:

$$\mathrm{seq}(\mathcal{Q}, b) = \mathrm{STR}(q_۱q_۲) \oplus \mathrm{STR}(q_۲q_۳) \oplus \ldots \oplus \mathrm{STR}(q_{r-۲}q_{r-۱}) \oplus \mathrm{STR}(b, q_{r-۱}q_r)$$

که در آن $\mathcal{Q} = \{q_۱, q_۲, \ldots, q_r\}$ است. توجه داشته باشید که $\mathrm{CS}(\mathrm{seq}(\mathcal{Q}, b)) = \mathrm{RM}(\mathrm{seq}(\mathcal{Q}, b))$. حال $\mathrm{OptHS}[i,j,b,l]$ را به این صورت تعریف می‌کنیم که حاوی تعداد میانبرهای کمینه $\mathcal{Q} = \{q_۱, \ldots, q_r\}$، از نقطه آغازین $p_۱$ و ($q_۱ = p_۱$) به میانبر پایانی p_ip_j، ($q_{r-۱} = p_i$ و $q_r = p_j$) بوده و $\mathrm{CS}(\mathrm{seq}(\mathcal{Q}, b))$ با اولین l عنصر از $\mathrm{CS}(\mathcal{P})$ تطابق داشته باشد. می‌توان به آسانی نشان داد که $\min_{i=۱}^{n} \mathrm{OptHS}[i, n, |\mathrm{STR}(p_ip_n)|, |\mathrm{CS}(\mathcal{P})|]$ برابر با تعداد میانبرهای ساده‌سازی بهینه هم‌فضا است، که در آن $|.|$ اندازه دنباله است.

برای پرکردن ماتریس OptHS، اغلب نیازداریم که بدانیم که CS از یک زیردنباله تهی است یا خیر. برای یک زیردنباله با اولین میانبر $p_{i_۱}p_{j_۱}$ و آخرین میانبر $p_{i_۲}p_{j_۲}$ و میانبرهای میانی $\ell_۱, \ldots, \ell_r$، ما اتصال آخرین a نماد از $\mathrm{STR}(p_{i_۱}p_{j_۱})$ و نمادهای دنباله‌ی $\ell_۱$ تا ℓ_r و اولین c نماد از $\mathrm{STR}(p_{i_۲}p_{j_۲})$ را توسط

$\text{seq}(i_۱, j_۱, i_۲, j_۲, a, c)$ و به صورت زیر نشان می‌دهیم:

$$\text{seq}(i_۱, j_۱, i_۲, j_۲, a, c) = \text{STR}(p_{i_۱}p_{j_۱}, a) \oplus \text{STR}(\ell_۱) \oplus \text{STR}(\ell_۲) \ldots \oplus \text{STR}(\ell_r) \oplus \text{STR}(c, p_{i_۲}p_{j_۲})$$

همچنین $\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c]$ را برابر تعداد میانبرهای زیردنباله‌ی کمینه قرار می‌دهیم که اولین و آخرین میانبر آن به ترتیب $p_{i_۱}p_{j_۱}$ و $p_{i_۲}p_{j_۲}$ هستند و $\text{CS}(\text{seq}(i_۱, j_۱, i_۲, j_۲, a, c))$ تهی است. توجه داشته باشید که :

$$\text{CS}(\text{seq}(i_۱, j_۱, i_۲, j_۲, a, c)) = \text{RM}(\text{seq}(i_۱, j_۱, i_۲, j_۲, a, c))$$

در ادامه شرح می‌دهیم که چگونه ماتریس‌های OptHS و OptNHS را پر می‌کنیم. ایده اصلی محاسبه $\text{OptHS}[i, j, b, l]$ و $\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c]$ به ترتیب زیر است: فرض کنید $\mathcal{Q}$ یک زیردنباله‌ی کمینه باشد که $\text{OptHS}[i, j, b, l]$ را مشخص کند. باید یک میانبر $p_{i'}p_{j'}$ در $\mathcal{Q}$ وجود داشته باشد به قسمی که برای برخی b'ها، $\text{STR}(p_{i'}p_{j'})[b'] = \text{CS}(\mathcal{P})[l]$ بوده و همچنین $\text{CS}(\text{seq}(i', j', i, j, |\text{STR}(p_{i'}p_{j'})| - b', b))$ تهی باشد (شکل ۵.۵ را ببینید). بنابراین

$$\text{OptHS}[i, j, b, l] = \min_{i' < j' \le i} (\text{OptHS}[i', j', b' - ۱, l - ۱] + \text{OptNHS}[i', j', i, j, |\text{STR}(p_{i'}p_{j'})| - b', b] - ۱)$$

که در آن min در بین کلیه میانبرهای $p_{i'}p_{j'}$ است که $\text{STR}(p_{i'}p_{j'})[b'] = \text{CS}(\mathcal{P})[l]$.

به روش مشابه، اگر $\mathcal{Q}$ یک زیردنباله‌ی کمینه باشد که $\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c]$ را مشخص کند، باید میانبر $p_{i'}p_{j'}$ی در زیردنباله‌ی کمینه‌ی $\mathcal{Q}$ وجود داشته باشد به طوری که

$$\text{STR}(p_{i'}p_{j'})[a'] = \text{STR}(p_{i_۱}p_{j_۱})[|\text{STR}(p_{i_۱}p_{j_۱})| - a + ۱]$$

بدین ترتیب اگر $\text{OptNHS}[i_۱, j_۱, i', j', a - ۱, a' - ۱]$ تهی باشد، $\text{STR}(p_{i'}p_{j'})[a']$ و $\text{STR}(p_{i_۱}p_{j_۱})[|\text{STR}(p_{i_۱}p_{j_۱})| - a + ۱]$ می‌توانند همدیگر را حذف نمایند (شکل ۶.۵ را ببینید). بنابراین

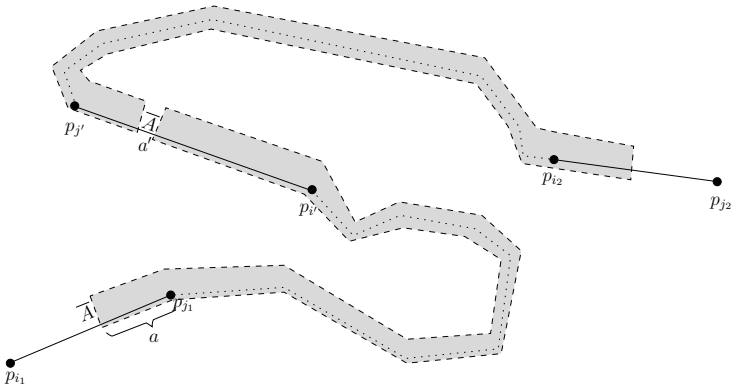
$$\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c] =$$

$$\min_{j_۱ \le i' < j' \le i_۲} (\text{OptNHS}[i_۱, j_۱, i', j', a, a' - ۱] + \text{OptNHS}[i', j', i_۲, j_۲, |\text{STR}(p_{i'}p_{j'})| - a', c] - ۱)$$

نحوه‌ی محاسبه‌ی دقیق $\text{OptHS}[i, j, b, l]$ و $\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c]$ در بخش بعدی آورده شده است.

۱.۲.۵ محاسبه‌ی OptHS و OptNHS

در این بخش نحوه‌ی دقیق محاسبه‌ی ماتریس‌های OptHS و OptNHS را ارایه می‌دهیم. توجه داشته باشید که ماتریس‌های OptHS و OptNHS، تعداد میانبرهای یک زیردنباله‌ی کمینه را نگهداری می‌کنند.



شکل ۶.۵: اگر $\mathcal{Q}$ یک زیرساده‌سازی باشد که $\text{OptNHS}[i_1, j_1, i_2, j_2, a, c]$ را مشخص کند یک میانبر $p_{i'}p_{j'}$ در زیرساده‌سازی $\mathcal{Q}$ وجود دارد به طوری که $\text{STR}(p_{i'}p_{j'})[a'] = \text{STR}(p_{i_1}p_{j_1})[|p_{i_1}p_{j_1}| - a + 1]$ که در آن a اندازه آخرین a عنصر از $\text{STR}(p_{i_1}p_{j_1})$ و a' شاخص $\overline{A}$ در میانبر $\text{STR}(p_{i'}p_{j'})$ است. بنابراین CS هر دو ناحیه خاکستری باید تهی باشد.

نحوه محاسبه‌ی OptHS

برای محاسبه‌ی $\text{OptHS}[i, j, b, l]$، که در آن $1 \le i < j \le n$، $0 \le b \le |\text{STR}(p_i p_j)|$ و $0 \le l \le |\text{CS}(\mathcal{P})|$ است، سه حالت زیر را داریم:

- $b = 0$:

$$\text{OptHS}[i, j, 0, l] = \min_{i'} \ \text{OptHS}[i', i, |\text{CS}(p_{i'}p_i)|, l] + 1 \text{ where } (i' < i)$$

- $l = 0$:

$$\text{OptHS}[i, j, b, 0] = \min_{i'} \ \text{OptNHS}[1, i', i, j, |\text{CS}(p_1 p_{i'})|, b] \text{ where } (i' \le i)$$

- $b, l > 0$:

$$\begin{aligned} \text{OptHS}[i, j, b, l] = \min_{i', j', b'} \ & \text{OptHS}[i', j', b' - 1, l - 1] + \\ & \text{OptNHS}[i', j', i, j, |\text{CS}(p_{i'}p_{j'})| - b' + 1, b] - 1 \\ \text{where} \ & ((i' = i, j' = j, b' = b - 1) \text{ or } (i' < j' \le i)) \text{ and} \\ & \text{CS}(\mathcal{P})[l] = \text{CS}(p_{i'}p_{j'})[b'] \end{aligned}$$

نحوه محاسبه‌ی OptNHS

برای پرکردن $\text{OptNHS}[i_1, j_1, i_2, j_2, a, c]$، که در آن $1 \le i_1 < j_1 \le i_2 < j_2 \le n$، $0 \le a \le |\text{STR}(p_{i_1}p_{j_1})|$ و $0 \le c \le |\text{CS}(p_{i_2}p_{j_2})|$ است، چهار حالت زیر را داریم:

- $c > ۰$:

$$\begin{aligned}\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c] = \min_{i', j', a'} \; & \text{OptNHS}[i_۱, j_۱, i', j', a, a' - ۱] + \\ & \text{OptNHS}[i', j', i_۲, j_۲, |\text{CS}(p_{i'} p_{j'})| - a', c - ۱] - ۱ \\ \text{where} \quad & (j_۱ \le i' < j' \le i_۲) \text{ and } (۰ < a' \le |\text{CS}(p_{i'} p_{j'})|) \\ & \text{and } \text{CS}(p_{i'} p_{j'})[a'] = \text{CS}(p_{i_۲} p_{j_۲})[c]\end{aligned}$$

- $a > ۰$ و $c = ۰$:

$$\begin{aligned}\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, ۰] = \min_{i', j', c'} \; & \text{OptNHS}[i_۱, j_۱, i', j', a - ۱, c' - ۱] + \\ & \text{OptNHS}[i', j', i_۲, j_۲, |\text{CS}(p_{i'} p_{j'})| - c', ۰] - ۱ \\ \text{where} \quad & (j_۱ \le i' < j' \le i_۲) \text{ and } (۰ < l' \le |\text{CS}(p_{i'} p_{j'})|) \\ & \text{and } \text{CS}(p_{i'} p_{j'})[a'] = \text{CS}(p_{i_۱} p_{j_۱})[|\text{CS}(p_{i_۱} p_{j_۱})| - a + ۱]\end{aligned}$$

- $a, c = ۰$ و $j_۱ \neq i_۲$:

$$\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, ۰, ۰] = ۳ \text{ if } \text{CS}(p_{j_۱} p_{i_۲}) = \text{ NULL and otherwise } +\infty$$

- $a, c = ۰$ و $j_۱ = i_۲$:

$$\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, ۰, ۰] = ۲ \text{ if } \text{CS}(p_{j_۱} p_{i_۲}) = \text{ NULL and otherwise } +\infty$$

۲.۲.۵ خروجی الگوریتم

اندازه زیرساده‌سازی هم‌فضای بهینه در OptHS نگه داشته می‌شود. در نگاه اول این‌گونه به نظر می‌رسد که برای گزارش زیرساده‌سازی هم‌فضای بهینه، نیاز به هزینه‌ی زمان و فضای بیشتری داریم. اما برای پیدا کردن پاسخ بهینه $\min_{i=۱}^{n} \text{OptHS}[i, n, |\text{STR}(p_i p_n)|, |\text{CS}(\mathcal{P})|]$ کافی است یک بار دیگر ماتریس را پیمایش کنیم. به این ترتیب، بدون آن که مرتبه‌ی فضا و زمان را افزایش دهیم، خروجی الگوریتم را به دست می‌آوریم.

۳.۲.۵ پیچیدگی زمان و حافظه

می‌دانیم بعدهای $\text{OptNHS}[i_۱, j_۱, i_۲, j_۲, a, c]$ در بازه‌های $۱ \le i_۱, i_۲, j_۱, j_۲ \le n$ و $۰ \le a, c \le m$ هستند. همچنین بعدهای $\text{OptHS}[i, j, b, l]$ در بازه‌های $۱ \le i, j \le n$ و $۰ \le b \le m$ و $۰ \le l \le mn$

هستند. بنابراین، حافظه‌ای که این دو ماتریس نیاز دارند از مرتبه‌ی $O(n^۴m^۲)$ می‌باشد. توجه داشته باشید که حداکثر اندازه‌ی STR یک میانبر می‌تواند m باشد و حداکثر اندازه‌ی $\mathrm{CS}(\mathcal{P})$ می‌تواند mn باشد. برای پرکردن هر درایه از هر یک از ماتریس‌ها ما باید کمینه را از میان $O(n^۲)$ میانبر $p_{i'}p_{j'}$ انتخاب کنیم. در نتیجه این کار نیاز به $O(n^۶m^۲)$ زمان دارد. توجه داشته باشید که پیدا کردن نماد $\mathrm{STR}(p_{i'}p_{j'})$ که با l-امین نماد $\mathrm{CS}(\mathcal{P})$ هم‌خوانی داشته باشد، نیاز به $O(m)$ زمان دارد. اما می‌توان با ساخت یک آرایه دو بعدی کمکی با اندازه‌ی $O(n^۲m)$، برای نگهداری محل هر نماد از STR هر میانبر، آن را در زمان $O(۱)$ انجام داد. بعد از محاسبه‌ی ماتریس‌ها، $\min_{i=۱}^{n} \mathrm{OptHS}[i, n, |\mathrm{STR}(p_ip_n)|, |\mathrm{CS}(\mathcal{P})|]$ را می‌توان در زمان $O(n)$ به دست آورد.

قضیه ۱.۲.۵. *یک مسیر چندضلعی $\mathcal{P}$ با اندازه n را در صفحه‌ای حاوی m نقطه به عنوان مانع را در نظر بگیرید. همچنین $T_F(n)$ را زمان لازم برای محاسبه‌ی G_ϵ تحت معیار خطا (تابع فاصله) F در حضور مانع‌ها نظر بگیرید. ساده‌سازی هم‌فضای بهینه را می‌توان در زمان $O(n^۶m^۲) + T_F(n)$ و فضای $O(n^۴m^۲)$ به دست آورد.*

۳.۵ الگوریتم مکاشفه‌ای ساده‌سازی هم‌فضای مسیرهای عمومی

الگوریتم‌های ارایه شده در دو بخش قبل دارای پاسخ بهینه بودند. لیکن الگوریتم بهینه برای مسیرهای عمومی دارای زمان اجرای بالایی است. در برخی از کاربردها، زمان عامل مهمتری نسبت به کیفیت خروجی است. در این بخش یک الگوریتم مکاشفه‌ای برای ساده‌سازی هم‌فضای مسیر ارایه می‌دهیم که کمینه بودن اندازه خروجی را تضمین نمی‌کند، ولی دارای زمان اجرای کمتر از مرتبه‌ی دو است. به عبارت دقیق‌تر ما الگوریتمی با مرتبه زمانی $O(n + m\log(n + m) + k)$، برای شناسایی میانبرهای هم‌فضا، که در آن n طول مسیر، m تعداد نقطه‌های مانع در صفحه و k تعداد میانبرهای هم‌فضا شناسایی شده هستند، ارایه می‌کنیم.

این روش را می‌توان تحت هر معیار خطای دلخواه استفاده نمود. برای مثال زمان اجرای این روش تحت معیار خطای هاسدورف از مرتبه‌ی $O((n+m\log(n+m)+|G_\epsilon|)$، که در آن $|G|$ برابر با حداکثر تعداد میانبرهای هم‌فضای شناسایی شده است. روشی که ما ارایه می‌دهیم به مانند روش دی‌برگ و همکارانش [۲۰، ۲۱] (بهترین نتیجه قبلی)، همیشه کمینه طول مسیر را پیدا نمی‌کند. ولی الگوریتمی که ما ارایه می‌کنیم نتیجه [۲۰، ۲۱] را از نظر زمانی بیش از $\log n$ مرتبه بهبود می‌بخشد. الگوریتم مکاشفه‌ای ما دارای دو مرحله کلی است. در مرحله اول، یا پیش پردازش، یک چندضلعی ساده $\Psi(\mathcal{P}, S)$ با نام ناحیه مجاز ساخته می‌شود. در مرحله دوم میانبرهای مجاز با استفاده از نتایج پیش‌پردازش انجام شده، شناسایی می‌شوند.

۱.۳.۵ پیش‌پردازش چندضلعی

در مرحله پیش‌پردازش، ناحیه مجاز $\Psi(\mathcal{P}, S)$ را می‌سازیم. مرحله پیش پردازش شامل عملیات زیر است:

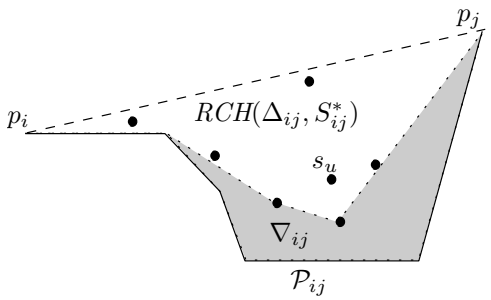
- تقسیم پوسته کوژ ساخته شده بر روی مسیر $\mathcal{P}$ به دو چندضلعی L و R.
- تقسیم چندضلعی L و R به چندضلعی‌های ساده Δ_{ij}.
- محاسبه *پوسته-کوژ- وابسته* برای نقطه‌های مانع موجود در Δ_{ij} و برخی نقطه‌های اضافه شده توسط الگوریتم ما.
- ساخت $\Psi(\mathcal{P}, S)$.

در ادامه هر مرحله را با جزئیات توضیح می‌دهیم. می‌دانیم که در گونه محدود ساده‌سازی مسیر، مسیر ساده شده $\mathcal{Q}$ باید زیرمجموعه‌ای از نقطه‌های $\mathcal{P}$ باشد. بنابراین کلیه میانبرهای q_iq_j، $1 \leq i < j \leq n$، در داخل پوسته کوژ مسیر $\mathcal{P}$ قرار می‌گیرند. از این پس، ما $CH(\mathcal{P})$ را پوسته کوژ مسیر $\mathcal{P}$ می‌نامیم. قبل از پرداختن به مرحله اول، کلیه نقطه‌های S که خارج از $CH(\mathcal{P})$ قرار می‌گیرند را از مجموعه نقطه‌های مانع حذف می‌کنیم.

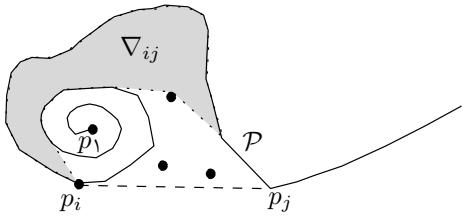
عموما داده ساختار پوسته کوژ به صورت یک مجموعه مرتب از نقطه‌ها است. حال فرض کنید که p_1 و p_n عضوی از نقطه‌های $CH(\mathcal{P})$ باشند. بعدا نحوه‌ی رفع این محدودیت را برای مسیرهای خاصی که یکی یا هردوی این نقطه‌ها در $CH(\mathcal{P})$ نباشند، را بیان می‌کنیم. مسیر چندضلعی $\mathcal{P}$ پوسته کوژ $CH(\mathcal{P})$ را به دو چندضلعی L و R تقسیم می‌کند. $CH(\mathcal{P})$ در نقطه‌ها p_1 و p_n به دو زنجیره $CH(\mathcal{P})_l$ و $CH(\mathcal{P})_r$ تقسیم می‌شود. محاسبه‌ای که در ادامه بر روی چندضلعی L انجام می‌دهیم، برای چندضلعی R نیز قابل اعمال است. بنابراین تنها محاسبه روی چندضلعی L را بیان می‌کنیم. زمان پردازش این مرحله، با توجه به آن که نیاز به محاسبه پوسته کوژ مسیر $\mathcal{P}$ داریم، $O(n \log n)$ است.

روشن است که بعضی از نقطه‌های $CH(\mathcal{P})$ و $\mathcal{P}$ مشترک هستند. بنابراین L لزوما یک چندضلعی ساده نیست. در مرحله دوم الگوریتم، L را به تعدادی چندضلعی Δ_{ij} تقسیم می‌کنیم. به ازای هر لبه از $CH(\mathcal{P})_l$ یک میانبر متناظر آن با نام p_ip_j وجود دارد. با ترکیب $\mathcal{P}(i, j)$ و میانبر p_ip_j چندضلعی Δ_{ij} را می‌سازیم. ساخت کلیه Δ_{ij}ها در زمان خطی قابل انجام است.

در مرحله سوم، ابتدا نقطه‌های S را در بین Δ_{ij}ها توزیع می‌کنیم. این کار را با پیش‌پردازش Δ_{ij}ها در زمان $O((n + m) \log n)$ و با استفاده از الگوریتم [۴۶] انجام می‌دهیم. حال فرض کنید مجموعه نقطه‌هایی از S که در چندضلعی Δ_{ij} جای می‌گیرند را با S_{ij} نشان دهیم. *پوسته-کوژ- وابسته* برای یک چندضلعی ساده X و مجموعه‌ای از نقطه‌های S درون X، عبارت است از کوتاه‌ترین حلقه داخل چندضلعی X که نقطه‌های



شکل ۷.۵: چندضلعی Δ_{ij} از لبه‌های زیرمسیر $\mathcal{P}(i,j)$ و میانبر p_ip_j در $CH(\mathcal{P})_l$ ساخته شده است. ناحیه سفید رنگ بین خطهای نقطه‌چین و خط‌چین پوسته-کوژ-وابسته S^*_{ij} در داخل Δ_{ij} است. ∇_{ij} با رنگ خاکستری نشان داده شده است.



شکل ۸.۵: نقطه p_1 عضوی از $CH(\mathcal{P})$ نیست. بنابراین به هنگام محاسبه پوسته-کوژ-وابسته در داخل Δ_{ij} ما $\mathcal{P}(1,i)$ را هم در نظر گرفته و $RCH(\Delta_{ij}, S^*_{ij} \cup \{p_1, p_2, ...p_{i-1}\})$ را حساب می‌کنیم. ناحیه خاکستری ∇_{ij} را نشان می‌دهد.

S را احاطه کرده باشد [۴۷]. از این به بعد پوسته-کوژ-وابسته یک چندضلعی ساده X و مجموعه نقطه‌های S را با $RCH(X,S)$ نشان می‌دهیم. حال پوسته-کوژ-وابسته چندضلعی Δ_{ij} و مجموعه نقطه‌های $S^*_{ij} = S_{ij} \cup \{p_i, p_j\}$ را حساب می‌کنیم. ما از الگوریتم ارایه شده توسط توسن [۴۷] برای محاسبه $RCH(\Delta_{ij}, S^*_{i,j})$ استفاده می‌کنیم. این روش بر روی چندضلعی ساده کار می‌کند. از آن جایی که Δ_{ij} یک چندضلعی ساده است، می‌توانیم از این روش استفاده کنیم. اگر S_{ij} خالی باشد، ما خروجی $RCH(\Delta_{ij}, S^*_{i,j})$ را برابر با Δ_{ij} قرار می‌دهیم. محاسبه $RCH(\Delta_{ij}, S^*_{i,j})$ برای همه Δ_{ij}ها و مجموعه نقطه‌های S در زمان $O((n+m)\log(n+m))$ قابل انجام است.

در مرحله چهارم $\Psi(\mathcal{P}, S)$ را می‌سازیم. ابتدا $\Psi(\mathcal{P}, S)_l$ را می‌سازیم. برای هر Δ_{ij} و $RCH(\Delta_{ij}, S^*_{i,j})$ ما چندضلعی ∇_{ij} را می‌سازیم (شکل ۷.۵ را ببینید):

$$\nabla_{ij} = \Delta_{ij} \backslash RCH(\Delta_{ij}, S^*_{i,j})$$

سپس اجتماع ∇_{ij}ها را به دست آورده و $\Psi(\mathcal{P}, S)_l$ را می‌سازیم. ما این مرحله‌ها را بر روی $CH(\mathcal{P})_r$ هم اجرا می‌کنیم. آنگاه $\Psi(\mathcal{P}, S)_l$ و $\Psi(\mathcal{P}, S)_r$ را ادغام می‌کنیم و $\Psi(\mathcal{P}, S)$ را می‌سازیم.

حال برمی‌گردیم به فرضی که برای نقطه‌های آغازین و پایانی کرده بودیم. در برخی حالت‌های خاص نقطه ابتدایی و یا انتهایی ممکن است روی $CH(\mathcal{P})$ قرار نگیرند. شکل ۸.۵ را ببینید که در آن ابتدای مسیر $\mathcal{P}$ شکلی حلقوی دارد. برای چنین حالت‌هایی به صورتی که در ادامه می‌آید عمل می‌کنیم. فرض کنید p_i اولین

نقطه بعد از p_1 در دنباله نقطه‌ها روی $CH(\mathcal{P})$ باشد و همچنین Δ_{ij} چندضلعی باشد که p_1 در آن واقع است. آنگاه قرار می‌دهیم:

$$\nabla_{ij} = \Delta_{ij} \backslash RCH(\Delta_{ij}, \{S^*_{ij} \cup \{p_1, p_2, ...p_{i-1}\})$$

بدین‌ترتیب ، زیرمجموعه‌ای از مسیر $\mathcal{P}$ را که داخل Δ_{ij} قرار می‌گیرد را حذف می‌کنیم. در چنین حالت‌هایی، این مرحله را اجرا نموده و $\Psi(\mathcal{P}, S)$ را به دست می‌آوریم. توجه داشته باشید که بعد از اجرای کل الگوریتم و پیدا کردن کمینه نقطه‌های مسیر $\mathcal{Q}$ باید دنباله نقطه‌های $\{p_1, p_2, ...p_{i-1}\}$ را به $\mathcal{Q}$ اضافه کنیم. در صورتی که نقطه انتهایی نیز حلقوی باشد، از مانند روش فوق عمل می‌کنیم. لم زیر را به آسانی می‌توان اثبات نمود.

لم ۱.۳.۵. *برای یک مسیر چندضلعی $\mathcal{P}$ داده شده و یک مجموعه از نقطه‌های مانع S، کلیه $s_i \in S$ بیرون از چندضلعی $\Psi(\mathcal{P}, S)$ که به روش فوق ساخته شده، قرار می‌گیرند.*

حال لم زیر را اثبات می‌کنیم.

لم ۲.۳.۵. *برای یک مسیر چندضلعی $\mathcal{P}$ داده شده و یک مجموعه از نقطه‌های مانع S، کلیه میانبرهای p_ip_j از $\mathcal{P}$ که داخل چندضلعی $\Psi(\mathcal{P}, S)$ قرار می‌گیرند هم‌فضا هستند.*

اثبات. از لم قبلی می‌دانیم که کلیه $s_i \in S$ بیرون $\Psi(\mathcal{P}, S)$ قرار می‌گیرد. در نتیجه هر میانبر p_ip_j واقع در $\Psi(\mathcal{P}, S)$ را می‌توان بدون آن که از روی نقطه sی عبور کنیم تبدیل به زیرمسیر $\mathcal{P}(i, j)$ نمود. بنابراین کلیه میانبرهایی که داخل $\Psi(\mathcal{P}, S)$ قرار می‌گیرند هم‌فضا هستند. □

توجه داشته باشید که برخی از میانبرهایی که بیرون $\Psi(\mathcal{P}, S)$ قرار می‌گیرند هم‌فضا هستند.

۲.۳.۵ شناسایی میانبرهای مجاز

حال که $\Psi(\mathcal{P}, S)$ را داریم، به شناسایی میانبرهای مجاز p_ip_j، $1 \leq i < j \leq n$ ، می‌پردازیم. ما یک میانبر p_ip_j را مجاز می‌خوانیم اگر داخل $\Psi(\mathcal{P}, S)$ قرار گیرد. به عبارت دیگر اگر p_ip_j با یک یا چند ضلع از ضلع‌های $\Psi(\mathcal{P}, S)$ در نقطه‌ای بجز p_i و p_j برخورد کند، آن میانبر مجاز نخواهد بود.

برای حل مسئله پیدا کردن میانبرهای مجاز می‌توانیم کلیه نقطه‌های برخورد میانبرها و ضلع‌های $\Psi(\mathcal{P}, S)$ را در زمان $O(n^2(n+m)\log(n+m))$ با روش ابتدایی به دست آوریم. برای حل کارای آن به این مسئله به عنوان مسئله قابلیت دید نگاه می‌کنیم. گوییم اگر یک نقطه p_j توسط نقطه p_i در داخل یک چندضلعی $\Psi(\mathcal{P}, S)$ دیده شود آنگاه p_ip_j یک میانبر معتبر است. به عبارت دیگر مسئله فوق به مسئله محاسبه گراف قابلیت دید تبدیل می‌شود. این مسئله از جمله مسایلی است که بسیار مورد مطالعه قرار گرفته است. بهترین

الگوریتم قبلی توسط بن- موشه [۴۳] ارایه شده است. الگوریتم آن‌ها یک چندضلعی و مجموعه‌ای از نقطه‌ها را به عنوان ورودی دریافت و جفت نقطه‌هایی که یکدیگر را می‌بینند را در زمان $O(n+m\log m\log(mn)+k)$ شناسایی می‌کند، که در آن n، m و k به ترتیب عبارتند از تعداد نقطه‌های چندضلعی، تعداد نقطه‌های داخل چندضلعی و تعداد نقطه‌هایی که یکدیگر را می‌بینند. تنها چیزی که باقی می‌ماند، پیچیدگی الگوریتم با توجه به شرایط مسئله ما است. تعداد نقطه‌ها در $\Psi(\mathcal{P},S)$ می‌تواند در بدترین حالت برابر $O(m+n)$ باشد، که در آن n تعداد نقطه‌های مسیر اصلی و m تعداد نقطه‌ها مانع در صفحه است. با اعمال این پارامترها در الگوریتم بن- موشه زمان $O(n+m\log m\log m(n+m)+k)$ را به دست می‌آوریم. توجه داشته باشید که در بدترین حالت k می‌تواند $O(n^2)$ باشد. این حالت فقط وقتی رخ می‌دهد که هیچ نقطه‌ای از نقطه‌های مجموعه S داخل $CH(\mathcal{P})$ قرار نگیرد. بنابراین، ما انتظار k بسیار کوچکتری را در کاربردهای واقعی داریم. به این ترتیب، نتیجه زیر را داریم:

قضیه ۳.۳.۵. *فرض کنید $\mathcal{P}$ یک مسیر چندضلعی عمومی باشد و S مجموعه‌ای شامل m نقطه به عنوان مانع روی صفحه باشد. الگوریتم مکاشفه‌ای وجود دارد که قادر به شناسایی زیرمجموعه‌ای از میانبرهای هم‌فضا p_ip_j در زمان $O((n+m)\log(n+m+T_F(n)))$ است، که در آن $T_F(n)$ زمان لازم برای محاسبه‌ی G_ϵ تحت معیار خطا (تابع فاصله) F در حضور مانع‌ها هست.*

۴.۵ جمع‌بندی

در این فصل یک چارچوب کلی برای ساده‌سازی هم‌فضای مسیرهای چندضلعی در حضور مانع‌ها ارایه نمودیم. با استفاده از این چارچوب می‌توان به ساده‌سازی هم‌فضا تحت معیار خطای دلخواه پرداخت. ما چهار الگوریتم برای این مسئله ارایه نمودیم: (آ) الگوریتم قویا- هم‌فضا برای مسیرهای x- یکنوا، (ب) الگوریتم قویا- هم‌فضا برای مسیرهای عمومی، (ج) الگوریتم بهینه هم‌فضا برای مسیرهای عمومی و (د) الگوریتم مکاشفه‌ای قویا- هم‌فضا برای مسیرهای عمومی. لازم به ذکر است که برای مسیرهای x- یکنوا، مسیر هم‌فضا حتما قویا- هم‌فضا نیز هست و در نتیجه الگوریتم قویا- هم‌فضای ارایه شده توسط ما، بهترین نتیجه برای مسیرهای هم‌فضا نیز هست.

جدول بعدی مقایسه‌ای است بین الگوریتم‌های ارایه شده در این بخش با بهترین نتیجه‌های قبلی:

جدول ۱.۵: مقایسه الگوریتم‌های ارایه شده برای ساده‌سازی قویا- هم‌فضا و هم‌فضا

نوع	نتیجه‌های قبلی		نتیجه‌های ما	
	زمان اجرا	حالت	زمان اجرا	حالت
مسیر x-یکنوا ،قویا-هم‌فضا	$O(n^2 + (m+n)\log(n+m))$ [۲۰]	مکاشفه‌ای	$O(m\log(m+n) + n\log n\log(n+m) + k) + T_F(n)$	اندازه پاسخ بهینه
مسیر عمومی، قویا-هم‌فضا	$O(n^2 + (m+n)\log(n+m))$ [۲۰]	مکاشفه‌ای	$O((n+m)\log(n+m) + k) + T_F(n)$	مکاشفه‌ای
مسیر عمومی، قویا-هم‌فضا	-	-	$O(n(m+n)\log(n+m)) + T_F(n)$	اندازه پاسخ بهینه
مسیر عمومی هم‌فضا	-	-	$O(n^6 m^2) + T_F(n)$	اندازه پاسخ بهینه

در جدول فوق n اندازه مسیر چندضلعی، m تعداد مانع‌های روی صفحه، k تعداد میانبرهای شناسایی شده (حداکثر n^2) و $T_F(n)$ زمان لازم برای محاسبه خطای میانبرهای مسیر تحت معیار خطای دلخواه F است. این زمان برای معیارهای خطای هاسدورف و فرشه به ترتیب از مرتبه‌ی $O(n^2)$ [۱۴] و $O(n^3)$ [۳۳] است.

فصل ۶

الگوریتم‌های ساده‌سازی تحت معیار قابلیت دید

در این بخش به ارایه نتیجه‌های به دست آمده ما در زمینه‌ی ساده‌سازی مسیر تحت معیار قابلیت دید می‌پردازیم. مسیر داده شده $\mathcal{P}$ واقع در چندضلعی ساده $\mathcal{R}$ را در نظر بگیرید. ما به دنبال یک مسیر ساده شده با طول کمینه تحت معیار قابلیت دید هستیم. ما مسئله را تحت دو معیار قابلیت دید ضعیف بررسی می‌کنیم: (آ) WVS و (ب) WVA. ابتدا ما به مسئله بیشینه خطای مسیر تحت معیار اندازه قابلیت دید ضعیف، $\mathrm{Err}^{\max}_{\mathrm{WVS}}$، می‌پردازیم. نشان می‌دهیم که با پیش پردازش چندضلعی $\mathcal{R}$ در زمان $O(m^{۷})$ و با استفاده از حافظه $O(m^{۶})$، می‌توان به پرسمان‌های مسیر دلخواه $\mathcal{P}$ در زمان $O(n(n+m))$ پاسخ داد (m اندازه چندضلعی و n اندازه مسیر است). سپس به مسئله جمع خطای مسیر تحت معیار مساحت قابلیت دید ضعیف، $\mathrm{Err}^{\mathrm{sum}}_{\mathrm{WVA}}$، می‌پردازیم. ابتدا نشان می‌دهیم که این مسئله ان‌پی-سخت است. سپس یک الگوریتم تقریبی برای آن ارایه می‌دهیم. این پژوهش با مشارکت آقای مجتبی نوری انجام شد. الگوریتم‌های ارایه شده فوق در قالب یک مقاله جمع‌بندی و برای ارسال به ژورنال آماده شده است:

1. S. Daneshpajouh, M. Nouri Byghi, M. Ghodsi, Computing Weak Visibility Line Simplification Inside a Simple Polygon.

۱.۶ ساده‌سازی تحت معیار بیشینه‌ی اندازه قابلیت دید

دراین بخش الگوریتمی برای مسئله‌ی #-کمینه، تحت معیار اندازه‌ی چندضلعی قابلیت دید، ارایه می‌دهیم. خطای مسیر ساده شده‌ی $\mathcal{Q}$ را $\max_{i=۱}^{k-۱} \mathrm{Err}_{\mathrm{WVS}}(q_iq_{i+۱})$ در نظر می‌گیریم. یادآوری می‌کنیم که WVS معیار خطای ساده‌سازی است و به صورت زیر تعریف می‌شود:

$$\mathrm{Err}_{\mathrm{WVS}}(p_ip_j) = \mathrm{WVP}(\mathcal{P}(i,j)) \ominus \mathrm{WVP}(p_ip_j)،$$

که در آن WVP چندضلعی قابلیت دید ضعیف، و $A \ominus B$ اندازه‌ی تفاضل بین دو چندضلعی A و B را مشخص می‌کند (یعنی $\mathrm{Err}_{\mathrm{WVS}}(p_ip_j)$ برابر است با تعداد ضلع‌هایی از چندضلعی که توسط مسیر $\mathcal{P}(i,j)$ دیده می‌شوند ولی توسط p_ip_j قابل دید نیستند). ما با سه زیرمسئله اصلی مواجه هستیم. سایر بخش‌ها با استفاده از چارچوب ایمایی و ایری [۱۰] قابل انجام است. درواقع پس از حل این سه زیر مسئله کافی است میانبرهایی که دارای خطای کمتر از ϵ هستند را به گراف G_ϵ اضافه کنیم و روش ایمایی و ایری را دنبال کنیم.

- یافتن WVP برای کلیه زیرمسیرهای $\mathcal{P}(i,j)$، $1 \le i < j \le n$.
- یافتن WVP برای کلیه میانبرهای p_ip_j، $1 \le i < j \le n$.
- محاسبه $\mathrm{Err}_{\mathrm{WVS}}$ برای کلیه میانبرهای p_ip_j، $1 \le i < j \le n$.

در ادامه ابتدا یک الگوریتم ابتدایی برای این سه زیرمسئله ارایه می‌دهیم. سپس یک الگوریتم کارا ارایه می‌دهیم، که با انجام پیش پردازش بر روی چندضلعی قادر است، به ازای هر پرسمان برای مسیر، در زمان بهتری پاسخ بهینه را محاسبه کند.

۱.۱.۶ الگوریتم ابتدایی

محاسبه‌ی $\mathrm{WVP}(\mathcal{P}(i,j))$ را می‌توان به صورت افزایشی و با استفاده از $\mathrm{WVP}(\mathcal{P}(i,j-1))$ در زمان $O(m)$ انجام داد. بنابراین WVP را برای کلیه زیرمسیرها می‌توان حداکثر در زمان $O(n^2m)$ به دست آورد. همچنین $\mathrm{WVP}(p_ip_j)$ را می‌توان در زمان $O(m)$ محاسبه نمود. بنابراین WVP را برای کلیه میانبرها می‌توان در زمان $O(n^2m)$ به دست آورد. برای هر میانبر محاسبه‌ی $\mathrm{Err}_{\mathrm{WVS}}(p_ip_j)$ را می‌توان در زمان $O(m)$ انجام داد. بنابراین $\mathrm{Err}_{\mathrm{WVS}}$ را برای کلیه میانبرها می‌توان در زمان $O(n^2m)$ به دست آورد. در نتیجه، زمان اجرای کل الگوریتم ساده‌سازی تحت معیار اندازه‌ی چندضلعی قابلیت دید از مرتبه‌ی $O(n^2m)$ خواهد بود.

ملاحظه ۱.۱.۶. *به آسانی می‌توان نشان داد که روش فوق بر روی مسئله* $\mathrm{Err}^{\max}_{\mathrm{WVA}}(\mathcal{Q})$ *نیز قابل اعمال است.*

در ادامه الگوریتمی ارایه می‌دهیم که هر یک از این زیرمسئله‌ها را در زمان بهتری حل می‌کند.

۲.۱.۶ محاسبه‌ی اندازه‌ی WVP برای کلیه زیرمسیرهای $\mathcal{P}(i,j)$

ابتدا، برای محاسبه‌ی اندازه WVP یک رابطه‌ی بازگشتی ارایه می‌دهیم. سپس از این رابطه استفاده نموده و الگوریتمی مبتنی بر برنامه‌ریزی پویا برای محاسبه‌ی اندازه چندضلعی قابلیت دید ضعیفِ همه زیرمسیرها ارایه می‌دهیم. ما پاره‌خط p_ip_{i+1} از مسیر $\mathcal{P}$ را با $e^{\mathcal{P}}_i$ نشان می‌دهیم. همچنین $e^{\mathcal{R}}_j$ را jامین ضلعِ چندضلعی $\mathcal{R}$

در نظر بگیرید. $\mathrm{WVPS}(\mathcal{P}(i,j))$ را مجموعه‌ای از ضلع‌های چندضلعی $\mathcal{R}$ قرار دهید که به صورت ضعیف توسط زیرمسیر $\mathcal{P}(i,j)$ قابل دید است. اندازه $\mathrm{WVP}(\mathcal{P}(i,j))$ را به صورت $|\mathrm{WVPS}(\mathcal{P}(i,j))|$ تعریف می‌کنیم، که در آن $|.|$ اندازه $\mathrm{WVPS}(\mathcal{P}(i,j))$ است. برای سادگی از $|\mathrm{WVP}(\mathcal{P}(i,j))|$ برای نشان دادن $|\mathrm{WVPS}(\mathcal{P}(i,j))|$ استفاده می‌کنیم. برای نمونه‌ای از WVP برای یک زیرمسیر $\mathcal{P}(i,j)$ شکل ۱.۶ب را ببینید. در این مثال $\mathrm{WVPS}(\mathcal{P}(i,j)) = \{e_1^{\mathcal{R}}, e_5^{\mathcal{R}}, \ldots, e_{19}^{\mathcal{R}}\}$ و اندازه‌ی $\mathrm{WVP}(\mathcal{P}(i,j))$ برابر است با ۱۶ ($|\mathrm{WVPS}(\mathcal{P}(i,j))| = 16$). به آسانی می‌توان دید که:

$$\begin{aligned}\mathrm{WVP}(\mathcal{P}(i,j)) = \quad & \mathrm{WVP}(\mathcal{P}(i-1,j)) \backslash \mathrm{WVP}(e_{i-1}^{\mathcal{P}}) \\ & \cup (\mathrm{WVP}(e_{i-1}^{\mathcal{P}}) \cap \mathrm{WVP}(\mathcal{P}(i,j)))\end{aligned}$$

بنابراین،

$$\begin{aligned}|\mathrm{WVP}(\mathcal{P}(i,j))| = \quad & |\mathrm{WVP}(\mathcal{P}(i-1,j))| - |\mathrm{WVP}(e_{i-1}^{\mathcal{P}})| \\ & + |(\mathrm{WVP}(e_{i-1}^{\mathcal{P}}) \cap \mathrm{WVP}(\mathcal{P}(i,j)))|\end{aligned}$$

مثال شکل ۱.۶ را ببینید. $\mathrm{Err}_{i,j,l}$ را مجموعه‌ای از لبه‌های $\mathcal{P}(i,j)$ قرار دهید که به صورت ضعیف توسط ضلع $e_l^{\mathcal{R}}$ قابل دید هستند. مقدار $X_{k,l} = 1$ قرار دهید اگر $e_l^{\mathcal{R}} \in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})$ توسط $e_k^{\mathcal{P}} \in \mathrm{Err}_{i,j,l}$ قابل دید است، که در آن $e_k^{\mathcal{P}}$، ضلع با کمترین مقدار شاخص است و در غیر این صورت $X_{k,l} = 0$ قرار دهید. برای مثال در شکل ۱.۶ $\mathrm{Err}_{i,j,18} = \{e_i^{\mathcal{P}}, \ldots, e_j^{\mathcal{P}}\}$ و $X_{i,16} = 1$ است. زیرا $e_{16}^{\mathcal{R}} \in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})$ توسط $e_i^{\mathcal{P}} \in \mathrm{Err}_{i,j,18}$ قابل دید است و $e_i^{\mathcal{P}}$ اولین ضلع $\mathrm{Err}_{i,j,18}$ است که $e_{18}^{\mathcal{R}}$ را می‌بیند. توجه داشته باشید که اگر $e_k^{\mathcal{P}}$ به صورت ضعیف $e_l^{\mathcal{P}}$ را می‌بیند ولی اولین ضلعی نیست که $e_l^{\mathcal{P}}$ را می‌بیند، آنگاه $X_{k,l} = 0$ است. بنابراین داریم:

$$|\mathrm{WVP}(\mathcal{P}(i,j))| = |\mathrm{WVP}(\mathcal{P}(i-1,j))| - |\mathrm{WVP}(e_{i-1}^{\mathcal{P}})| + \sum_{\substack{i \le k \le j \\ e_l^{\mathcal{R}} \in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})}} X_{k,l}$$

مشاهده کنید که $\sum_{\substack{i \le k \le j \\ e_l^{\mathcal{R}} \in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})}} X_{k,l}$ برابر است با اندازه $\mathrm{WVP}(e_{i-1}^{\mathcal{P}}) \cap \mathrm{WVP}(\mathcal{P}(i,j))$.
می‌توانیم عبارات بالا را به صورت زیر بازنویسی کنیم:

$$|\mathrm{WVP}(\mathcal{P}(i,j))| = |\mathrm{WVP}(\mathcal{P}(i-1,j))| - |\mathrm{WVP}(e_{i-1}^{\mathcal{P}})| + Y(i,j) \quad (۱.۶)$$،

که در آن

$$Y(i,j) = \sum_{\substack{i \le k \le j \\ e_l^{\mathcal{R}} \in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})}} X_{k,l}$$

حال می‌توانیم $Y(i,j)$ را به صورت یک رابطه بازگشتی بنویسم:

$$\begin{aligned} Y(i,j) &= \sum_{i\le k\le j} \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{k,l} \\ &= \sum_{i\le k\le j-1} \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{k,l} + \sum_{k=j} \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{k,l} \\ &= Y(i,j-1) + \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{j,l} \end{aligned} \qquad (۲.۶)$$

با داشتن رابطه‌ی (۱.۶) و (۲.۶) و با استفاده از برنامه‌ریزی پویا، ما راه‌حل بهینه برای حل مسئله محاسبه‌ی اندازه WVP برای همه‌ی زیرمسیرهای $\mathcal{P}$ را ارایه می‌دهیم. برای i,j با $1 \le i < j \le n$، مقدار $\mathcal{D}[i,j]$ را برابر با اندازه $\mathrm{WVP}(\mathcal{P}(i,j))$ قرار می‌دهیم. با استفاده از رابطه (۱.۶)، ما $\mathcal{D}[i,j]$ را به صورت بازگشتی به روش زیر محاسبه می‌کنیم:

- $i = 1$: $\mathcal{D}[1,j] \leftarrow$ the size of $\mathrm{WVP}(\mathcal{P}(1,j)), 1 \le j \le m-1$.
- $i > 1$: $\mathcal{D}[i,j] \leftarrow \mathcal{D}[i-1,j] - \mathcal{D}[i-1,i] + Y[i,j], i \le j \le m-1$.

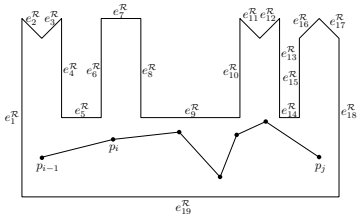
که در آن ماتریس $Y[i,j]$ تعداد ضلع‌های قابلیت دیدِ $\mathrm{WVP}(e_{i-1}^{\mathcal{P}})$ توسط ضلع‌های $\mathcal{P}(i,j)$ را ذخیره می‌کند. توجه داشته باشید که $0 \le Y[i,j] \le |\mathrm{WVP}(e_{i-1}^{\mathcal{P}})| \le |\mathcal{R}| = m$ است. بار دیگر، از برنامه‌ریزی پویا استفاده می‌کنیم و $Y[i,j]$ را با استفاده از رابطه (۲.۶) به صورت بازگشتی، و به صورت زیر محاسبه می کنیم:

- $i = 1$: $Y[1,j] \leftarrow \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{j,l}, 1 \le j \le m-1$.
- $i > 1$: $Y[i,j] \leftarrow Y[i,j-1] + \sum_{e_l^{\mathcal{R}}\in \mathrm{WVP}(e_{i-1}^{\mathcal{P}})} X_{j,l}, 1 \le j \le m-1$.

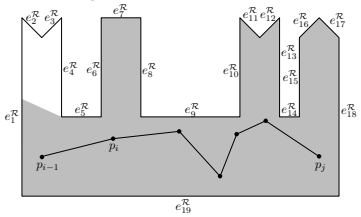
حال مسئله این خواهد بود که به طریقی $\mathcal{D}$ و $Y[i,j]$ را محاسبه نماییم که کل زمان اجرا از $O(n(n+m))$ بیشتر نشود. در بخش بعدی شبه‌کد برای الگوریتم پویا را ارایه می دهیم.

شبه کد برای الگوریتم برنامه‌ریزی پویا

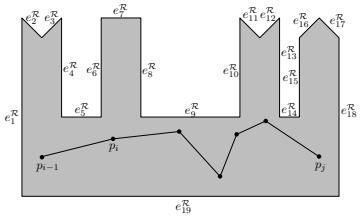
الگوریتم ۲.۶ تابع اصلی برای محاسبه تابع پویای ارایه شده است. در خط ۱ از این الگوریتم، تابع FillMatrix فراخوانی می‌شود (الگوریتم ۳.۶). در تابع FillMatrix ابتدا ضلع‌هایی از $\mathcal{R}$ را که توسط $e_i^{\mathcal{P}}$، $1 \le i < n$، قابل دیدن هستند را پیدا می‌کنیم و آن‌ها را در ماتریس دوبعدی $\mathrm{VM}[1..n-1, 1..m-1]$ قرار می‌دهیم. هر خانه از ماتریس $\mathrm{VM}[i,j]$ دارای دو متغیر *isVisible* و *next* است. اگر $e_i^{\mathcal{P}}$ ضلع $e_j(\mathcal{R})$ را ببیند، آنگاه مقدار *isVisible* را برابر true قرار می‌دهیم، در غیر این صورت مقدار آن را برابر false قرار می‌دهیم. ما از



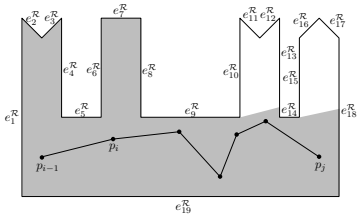
(آ) زیر-مسیر $\mathcal{P}(i-۱,j)$ در داخل چندضلعی ساده $\mathcal{R}$.



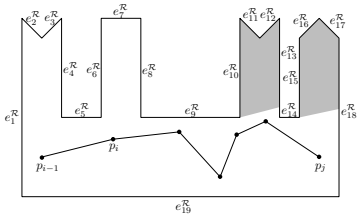
(ب) $\mathrm{WVP}(\mathcal{P}(i,j))$ با رنگ خاکستری نشان داده شده است و اندازه آن برابر است با $|\mathrm{WVPS}(\mathcal{P}(i,j))| = |\{e^{\mathcal{R}}_{۱}, e^{\mathcal{R}}_{۵}, \ldots, e^{\mathcal{R}}_{۱۹}\}| = ۱۶$.



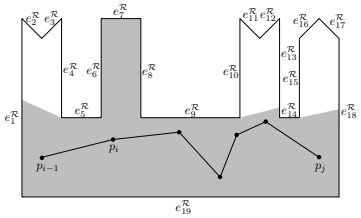
(ج) $\mathrm{WVP}(\mathcal{P}(i-۱,j))$ با رنگ خاکستری نشان داده شده است و $|\mathrm{WVPS}(\mathcal{P}(i-۱,j))| = |\{e^{\mathcal{R}}_{۱}, e^{\mathcal{R}}_{۲}, \ldots, e^{\mathcal{R}}_{۱۹}\}| = ۱۹$ است.



(د) $\mathrm{WVP}(e^{\mathcal{P}}_{i-۱})$ با رنگ خاکستری نشان داده شده است و $|\mathrm{WVP}(e^{\mathcal{P}}_{i-۱})| = |\{e^{\mathcal{R}}_{۱}, \ldots, e^{\mathcal{R}}_{۹}, e^{\mathcal{R}}_{۱۳}, e^{\mathcal{R}}_{۱۴}, e^{\mathcal{R}}_{۱۸}, e^{\mathcal{R}}_{۱۹}\}| = ۱۳$ است.



(ه) $\mathrm{WVP}(\mathcal{P}(i-۱,j))\backslash\mathrm{WVP}(e^{\mathcal{P}}_{i-۱})$ خاکستری است.



(و) $|\mathrm{WVP}(e^{\mathcal{P}}_{i-۱}) \cap \mathrm{WVP}(\mathcal{P}(i,j))| = |\{e^{\mathcal{R}}_{۱}, e^{\mathcal{R}}_{۵}, e^{\mathcal{R}}_{۶}, e^{\mathcal{R}}_{۷}, e^{\mathcal{R}}_{۸}, e^{\mathcal{R}}_{۹}, e^{\mathcal{R}}_{۱۳}, e^{\mathcal{R}}_{۱۴}, e^{\mathcal{R}}_{۱۸}, e^{\mathcal{R}}_{۱۹}\}|$ و اندازه آن برابر با ۱۰ است.

شکل ۱.۶: زیرمسیر $\mathcal{P}(i-۱,j)$ واقع در چندضلعی $\mathcal{P}$. همان گونه که در شکل مشخص است $|\mathrm{WVP}(\mathcal{P}(i,j))| = ۱۶$ است و برابر است با $|\mathrm{WVP}(\mathcal{P}(i-۱,j))| - |\mathrm{WVP}(e^{\mathcal{P}}_{i-۱})| + |(\mathrm{WVP}(e^{\mathcal{P}}_{i-۱}) \cap \mathrm{WVP}(\mathcal{P}(i,j)))| = ۱۹ - ۱۳ + ۱۰ = ۱۶$.

الگوریتم ۴.۶ برای پرکردن متغیر *next* استفاده می‌کنیم (خط ۲ از الگوریتم ۲.۶). در این تابع، ما شاخص اولین ضلع پس از $e_i^{\mathcal{P}}$ را که بتواند $e_j(\mathcal{R})$ را ببیند پیدا نموده و سپس $\mathrm{VM}[i,j].\mathit{next}$ را برابر شاخص آن قرار می‌دهیم. به عبارت دقیق‌تر، برای پرکردن $\mathrm{VM}[i,j].\mathit{next}$، برای یک ضلع خاص j از مسیر $\mathcal{R}$، از آخرین ضلع $\mathcal{P}$ به سمت اولین ضلع حرکت می‌کنیم (از $i=n$ به ۱)، و مقدار $\mathrm{VM}[i,j].\mathit{next}$ را برابر با نزدیکترین ضلع i'ی، $i'>i$، قرار می‌دهیم به شرط آن که $\mathrm{VM}[i',j].\mathit{isVisible}$ برابر true باشد.

حال نشان می‌دهیم که چگونه ماتریس $Y[1..n, 1..n]$ می‌تواند با استفاده از ماتریس VM پر شود. ما نحوه محاسبه مرحله iام را شرح می‌دهیم. سایر مرحله‌ها را به راحتی می‌توان در خطهای ۳ تا ۱۱ از الگوریتم ۲.۶ دنبال نمود. در خط ۴ از الگوریتم ۲.۶، تابع getNextEdgesList را فراخوانی می‌کنیم. این تابع یک شاخص i که نماینده ضلع $e_i^{\mathcal{P}}$ است را به عنوان ورودی دریافت نموده و برای هر ضلع $e_j^{\mathcal{R}}$ در $\mathrm{WVP}(e_i^{\mathcal{P}})$، اولین ضلع پس از $e_i^{\mathcal{P}}$ را پیدا می‌کند که به صورت ضعیف $e_j^{\mathcal{R}}$ را می‌بیند و آن را به یک فهرست از ضلع‌های S اضافه می‌کند. این تابع این فهرست را به عنوان خروجی باز می‌گرداند. توجه داشته باشید که یک ضلع $e^{\mathcal{P}}$ ممکن است چندین بار در فهرست S ظاهر شود. در خط ۷ از الگوریتم ۲.۶، ما به طریق کارا $i+1 \le j \le m-1$، $\sum_{e_l^{\mathcal{R}} \in \mathrm{WVP}(e_i^{\mathcal{P}})} X_{j,l}$ را محاسبه می‌کنیم و آن را در آرایه $L[1\ldots n]$ قرار می‌دهیم ($L[j] = \sum_{e_l^{\mathcal{R}} \in \mathrm{WVP}(e_i^{\mathcal{P}})} X_{j,l}$). سپس، در خطهای ۷ تا ۹، مطابق با تابع بازگشتی ارایه شده، ماتریس Y را بروزرسانی می‌کنیم.

وقتی کل ماتریس Y پر شد، محاسبه ماتریس $\mathcal{D}$ آسان خواهد بود. زیرا تنها به اعمال تابع پویای ارایه شده نیاز داریم. جزئیات این محاسبه در خطهای ۱۲ تا ۱۳ از الگوریتم ۲.۶ آمده است.

وقتی ماتریس $\mathcal{D}$ پر شد، آنگاه اندازه چندضلعی قابلیت دید ضعیفِ $\mathcal{P}(i,j)$ برابر با مقدار $\mathcal{D}[i,j]$ خواهد بود.

تحلیل

تابع FillMatrix، الگوریتم ۳.۶، در زمان $O(n(n+m))$ اجرا می‌شود. توجه داشته باشید که خط ۳ از این الگوریتم، $O(m)$ زمان طول می‌کشد. همچنین، FillNextEdges، الگوریتم ۴.۶، دارای $O(nm)$ زمان است. بنابراین دوخط اول الگوریتم ۲.۶ در زمان $O(n(n+m))$ اجرا می‌شود.

تابع getNextEdgesList، الگوریتم ۵.۶، در زمان $O(m)$ اجرا می‌شود. بنابراین، برای یک i مشخص، خط ۴ از الگوریتم ۲.۶ $O(m)$ زمان طول می‌کشد. همچنین برای یک i معین، خطهای ۶ و ۷ $O(m)$ زمان طول می‌کشند. توجه داشته باشید که اندازه آرایه S حداکثر برابر با m است. خطهای ۸ تا ۱۱ $O(n)$ زمان طول می‌کشند. بنابراین محاسبه ماتریس Y، خطهای ۳ تا ۱۱، می‌تواند در زمان $O(n(n+m))$ انجام شود.

ماتریس $\mathcal{D}$ می‌تواند در زمان $O(n^2)$، اجرا شود (خطهای ۱۲ تا ۱۶ از الگوریتم ۲.۶). درنتیجه، کل الگوریتم $O(n(n+m))$ طول خواهد کشید. بنابراین لم زیر را داریم.

Algorithm ***ComputeWeakVisibilitySizeSubPath***
Input: n: number of edges of $\mathcal{P}$
Output: $\mathcal{D}[1..n-1, 1..n-1]$
1. FillMatrix(VM)
2. FillNextEdges(VM)
3. **for** $i = 1$ to $n-1$
4. **do** $S \leftarrow$ getNextEdgesList$(i-1)$
5. $L[k] \leftarrow 0, 1 \leq k < n$
6. **for** $k = 1$ to Size(S)
7. $L[S[k]] \leftarrow L[S[k]] + 1$
8. **for** $j = i$ to $n-1$
9. **do if** i=1
10. **then** $Y[i,j] = L[j]$
11. **else** $Y[i,j] = Y[i,j-1] + L[j]$
12. **for** $j = 1$ to $n-1$
13. **do** $\mathcal{D}[1,j] \leftarrow$size of WVP$(\mathcal{P}(1,j))$
14. **for** $i = 2$ to $n-1$
15. **do for** $j = i$ to $n-1$
16. **do** $\mathcal{D}[i,j] \leftarrow \mathcal{D}[i-1,j] - \mathcal{D}[i-1,i] + Y[i,j]$
17. **return** $\mathcal{D}$

شکل ۲.۶: شبه کد ComputeWeakVisibilitySizeSubPath

لم ۲.۱.۶. *برای یک مسیر چندضلعی داده شده $\mathcal{P}$ به طول n واقع در چندضلعی ساده $\mathcal{R}$ با اندازه m، می‌توان اندازه* WVP *را برای همه $O(n^2)$ زیرمسیر $\mathcal{P}(i,j)$، $1 \leq i < j \leq n$، در زمان $O(n(n+m))$ محاسبه نمود.*

۳.۱.۶ محاسبه‌ی اندازه‌ی چندضلعی قابلیت دید ضعیف برای یک میانبر

می‌دانیم که WVP یک پاره‌خط در داخل چند ضلعی با اندازه m را در زمان $O(m)$ می‌توان محاسبه نمود. بنابراین، $O(n^2 m)$ زمان برای پاسخ به $O(n^2)$ پرسمان برای چنین پاره‌خط‌هایی لازم است. در ادامه، الگوریتمی را ارایه می‌دهیم که پاسخ به پرسمان $O(n^2)$ میانبر را در مجموع در زمان $O(n^2)$ انجام می‌دهد. این الگوریتم ابتدا به پیش پردازش چندضلعی پرداخته و سپس قادر است به چنین پرسمان‌هایی سریعا پاسخ دهد. در ادامه لم زیر را اثبات می‌کنیم.

لم ۳.۱.۶. *اگر دو سرِ دوپاره‌خط در داخل یک چندضلعی ساده، در داخل یک ناحیه قابلیت دید قرار گیرد، آنگاه اندازه چندضلعی قابلیت دید ضعیف آن‌ها، یکسان است.*

اثبات. s_i و s'_i دو نقطه در داخل یک ناحیه قابلیت دید S_i از یک بخش‌بندی ناحیه قابلیت دید برای $\mathcal{R}$ را در نظر بگیرید. همچنین، s_j و s'_j را دو نقطه در S_j در نظر بگیرید. ادعا می‌کنیم که دو پاره‌خط $s_i s_j$ و

```
Algorithm FillMatrix
Input: VM: matrix VM
Output: VM: matrix VM
1.  for i = 1 to n
2.      do WVPArray: matrix[1..m]
3.         WVPArray ←ComputeWVP(e_i^P)
4.         for j = 1 to n
5.             do VM[i, j].isVisible=false
6.                if (WVPArray[j] = true)
7.                  then VM[i, j].isVisible=true
8.  return VM
```

شکل ۳.۶: شبه کد FillMatrix

```
Algorithm FillNextEdges
Input: VM: matrix VM
Output: VM: matrix VM
1.  for j = 1 to m
2.      do NextIndex ←null
3.         for i = n to 1
4.             do if (VM[i, j].isVisible=true)
5.                  then VM[i, j].next ←NextIndex
6.                       NextIndex ←i
7.  return VM
```

شکل ۴.۶: شبه کد FillNextEdges

$s'_is'_j$ دارای دو چندضلعی قابلیت دید با اندازه یکسان هستند. توجه داشته باشید که s_is_j و $s'_is'_j$ ممکن است ناحیه‌های قابلیت دید متفاوتی را قطع نمایند (شکل ۴.۲ را ببینید). با توجه به آن که s_i و s'_i در یک ناحیه قابلیت دید واقع شده‌اند و آن‌ها یک دنباله از نقطه‌ها و ضلع‌ها از $\mathcal{R}$ را می‌بینند، آنگاه $\mathrm{SPT}(s_i)$ با $\mathrm{SPT}(s'_i)$ برابر خواهد بود، که در آن $\mathrm{SPT}(s)$ درخت کوتاه‌ترین مسیر در $\mathcal{R}$ با ریشه s است. استدلال مشابه برای s_j و s'_j نیز صادق است. می‌دانیم که الگوریتم گویباس و همکارانش [۳۹] برای محاسبه چندضلعی قابلیت دید ضعیف یک پاره‌خط s_is_j در داخل چندضلعی $\mathcal{R}$، مبتنی بر $\mathrm{SPT}(s_i)$ و $\mathrm{SPT}(s_j)$ است. با توجه به آن که $\mathrm{SPT}(s_i) = \mathrm{SPT}(s'_i)$ و $\mathrm{SPT}(s_j) = \mathrm{SPT}(s'_j)$، می‌توانیم نتیجه‌گیری نماییم که الگوریتم [۳۹] مجموعه ضلع‌های یکسانی از چندضلعی را به عنوان $\mathrm{WVPS}(s_is_j)$ و $\mathrm{WVPS}(s'_is'_j)$ باز می‌گرداند. این بدان معنی است که $\mathrm{WVPS}(s_is_j) = \mathrm{WVPS}(s'_is'_j)$. بنابراین، $|\mathrm{WVP}(s_is_j)| = |\mathrm{WVP}(s'_is'_j)|$. □

پیش‌پردازش چندضلعی $\mathcal{R}$

ما چندضلعی $\mathcal{R}$ را به ناحیه‌های قابلیت دید تقسیم می‌کنیم، به گونه‌ای که برای هر ناحیه، دنباله یکسانی از نقطه‌ها و ضلع‌های چندضلعی $\mathcal{R}$ از هر نقطه در داخل آن ناحیه قابل دید باشند. S_i را ناحیه iام در نظر بگیرید.

Algorithm ***getNextEdgesList***
Input: i: index of an edge of $\mathcal{P}$
Output: S: List of edges
1. $S \leftarrow \emptyset$
2. **for** $j = 1$ to $m - 1$
3. **do if** (VM$[i, j]$*isVisible* = true)
4. **then if** (VM$[i, j]$.*next* is not null)
5. **then** Add VM$[i, j]$.*next* to S
6. **return** S

شکل ۵.۶: شبه کد getNextEdgesList

دو ناحیه S_i و S_j یکدیگر را می‌بینند اگر یک نقطه s_i در S_i و یک نقطه s_j در S_j وجود داشته باشند به گونه‌ای که s_i و s_j یکدیگر را ببینند. ما می‌توانیم بررسی کنیم که آیا دو ناحیه قابلیت دید یکدیگر را می‌بینند یا خیر، و اگر می‌بینند چنین زوج (s_i, s_j) را در زمان $O(m)$ پیدا کنیم. برای هر جفت (s_i, s_j)، نقطه s_i را به نقطه s_j با یک پاره‌خط وصل می‌کنیم و آن را به صورت s_is_j نشان می‌دهیم. برای هر s_is_j، ما $\mathrm{WVP}(s_is_j)$ را در زمان $O(m)$ محاسبه می‌کنیم و اندازه چندضلعی قابلیت دید آن را به دست می‌آوریم. بنابراین، اندازه چندضلعی قابلیت دید کلیه چنین پاره‌خط‌هایی در زمان $O(m^۷)$ قابل محاسبه است. ما اندازه WVP را، برای همه s_is_j در یک جدول $\mathcal{T}[۱ \ldots m^۳, ۱ \ldots m^۳]$ با اندازه $O(m^۶)$ قرار می‌دهیم.

مرحله پرسمان

حال، برای یک مسیر چندضلعی داده شده $\mathcal{P}$ واقع در چندضلعی پیش پردازش شده $\mathcal{R}$، ما اندازه چندضلعی قابلیت دید میانبرهای آن را محاسبه می‌کنیم. برای مسیر داده شده، ما ابتدا، به مکان یابی هر نقطه $p_i \in \mathcal{P}$ می‌پردازیم و شاخص ناحیه قابلیت دید آن را در زمان $O(\log m)$ پیدا می‌کنیم. به این ترتیب، پردازش برای کلیه نقطه‌های مسیر در زمان $O(n \log m)$ قابل انجام خواهد بود. حال برای هر میانبر p_ip_j، $۱ \leq i < j \leq n$، در $\mathcal{R}$، اندازه $\mathrm{WVP}(p_ip_j)$ را با استفاده از جدول $\mathcal{T}$ پیدا می‌کنیم. با توجه به آن که شاخص p_i و p_j را داریم، این پردازش در زمان $O(۱)$ انجام می‌شود.

بنابراین، با توجه به لم ۳.۱.۶، لم زیر را خواهیم داشت.

لم ۴.۱.۶. *با استفاده از پیش پردازش با زمان $O(m^۷)$ بر روی یک چندضلعی ساده $\mathcal{R}$ و به کارگیری حافظه با اندازه $O(m^۶)$، می‌توان اندازه WVP را برای همه میانبرهای p_ip_j از $\mathcal{P}$، $۱ \leq i < j \leq n$، در زمان $O(n^۲)$ محاسبه نمود.*

۴.۱.۶ محاسبه‌ی خطای کلیه میانبرها

ابتدا به نحوه‌ی محاسبه‌ی خطای میانبرها می‌پردازیم. برای این محاسبه از لم زیر استفاده می‌کنیم.

لم ۵.۱.۶. *برای یک زیرمسیر $\mathcal{P}(i,j)$ واقع در یک چندضلعی ساده، $\mathrm{WVP}(p_ip_j) \subseteq \mathrm{WVP}(\mathcal{P}(i,j))$ است.*

اثبات. اگر چندضلعی بدون حفره و ساده باشد، به سادگی می‌توان نشان داد که یک میانبر p_ip_j نمی‌تواند بخشی از چندضلعی را ببینید به طوری که زیرمسیر متناظرش ($\mathcal{P}(i,j)$) آن بخش را نبیند. □

با توجه به لم ۵.۱.۶ برای محاسبه‌ی خطای یک میانبر خواهیم داشت:

$$\begin{aligned}\mathrm{Err}_{\mathrm{WVS}}(p_ip_j) = &\quad |\mathrm{WVP}(\mathcal{P}(i,j)\backslash \mathrm{WVP}(p_ip_j)| \\ = &\quad |\mathrm{WVP}(\mathcal{P}(i,j)| - |\mathrm{WVP}(p_ip_j)|\end{aligned}$$

بنابراین خواهیم داشت.

لم ۶.۱.۶. *برای یک مسیر $\mathcal{P}$ واقع در یک چندضلعی ساده، خطای کلیه $O(n^2)$ میانبر از مسیر را، با فرض داشتن اندازه‌ی WVP کلیه میانبرها و اندازه‌ی WVP کلیه زیرمسیرها، می‌توان در زمان $O(n^2)$ محاسبه نمود.*

در ادامه نحوه‌ی محاسبه‌ی اندازه‌ی WVP برای کلیه میانبرها و کلیه زیرمسیرها را نشان می‌دهیم.
بنابراین قضیه زیر را خواهیم داشت.

قضیه ۷.۱.۶. *با انجام پیش‌پردازش بر روی $\mathcal{R}$ در زمان $O(m^7)$ و با استفاده از حافظه $O(m^6)$، می‌توان مسئله ساده‌سازی #-کمینه، با خطای $\mathrm{Err}^{\max}_{\mathrm{WVS}}$، را برای یک پرسمان مسیر $\mathcal{P}$ با اندازه $O(n)$ در $O(n(n+m))$ زمان حل نمود.*

۲.۶ ساده‌سازی تحت معیار مجموع مساحت قابلیت دید

در این بخش به مسئله‌ی ساده‌سازی مسیر تحت معیار مجموع مساحت قابلیت دید ($\mathrm{Err}^{\mathrm{sum}}_{\mathrm{WVA}}(\mathcal{Q})$) می‌پردازیم. ابتدا نشان می‌دهیم این مسئله برای مجموع خطای مسیر به طوری که مجموع خطای مسیر بهینه، برابر C باشد، یک مسئله‌ی ان‌پی سخت است. سپس یک الگوریتم تقریبی با زمان چند جمله‌ای برای آن ارایه می‌دهیم. برای تعریف خطای $\mathrm{Err}_{\mathrm{WVA}}$ به بخش ۹.۲ مراجعه نمایید.

۱.۲.۶ اثبات ان‌پی- سخت بودن مسئله

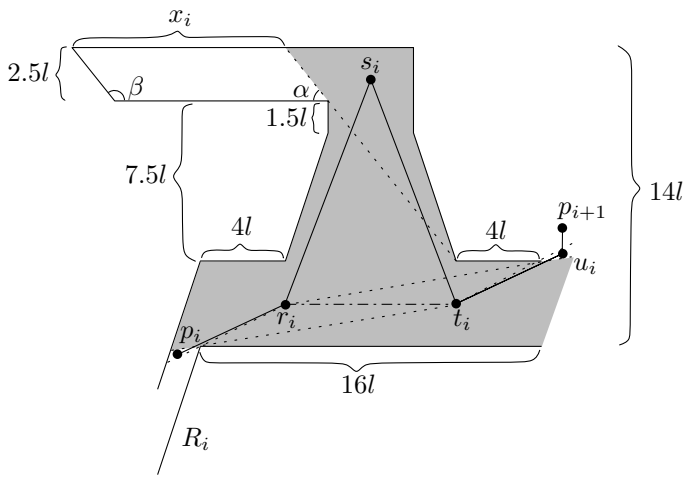
می‌دانیم مسئله‌ی مجموعِ زیرمجموعه[۱] یک مسئله ی ان‌پی- تمام است. در این مسئله یک مجموعه‌ی مثبت از اعداد $A = \{a_۱, \ldots, a_n\}$ و یک عدد صحیح مثبت C، که جمع یا مجموع نامیده می‌شود، داده شده است و هدف یافتن یک زیرمجموعه‌ی A' از A است به طوری که $\sum_{a_i \in A'} a_i = C$ باشد.

قضیه ۱.۲.۶. *برای یک مسیر چندضلعی داده شده $\mathcal{P}$ واقع در داخل چندضلعی ساده‌ی $\mathcal{R}$، و یک عدد مثبت k و یک هزینه‌ی C، مسئله‌ی یافتن یک ساده‌سازی $\mathcal{Q}$ با حداکثر k پاره‌خط و مجموع خطای تقریب C تحت معیار مساحت قابلیت دید ضعیف، ان‌پی- سخت است.*

اثبات. اثبات یک کاهش از مسئله‌ی مجموع زیرمجموعه است. مجموعه‌ی $A = \{a_۱, \ldots, a_n\}$ را یک نمونه از مسئله‌ی مجموع زیرمجموعه در نظر بگیرید. همچنین یک مقدار مثبت C داده شده است. برای هر $a_i \in A$، $\mathcal{P}(i)$ و $\mathcal{R}(i)$ را به ترتیب زیرمسیر از p_i به $p_{i+۱}$ و چندضلعی اطراف این زیرمسیر در نظر بگیرید، تصویر آن در شکل ۶.۶ آمده است. مختصات دقیق نقطه‌ها نیز در آن مشخص شده است. در شکل فوق l را برابر با واحد در نظر بگیرید. همچنین طول x_i را برابر $a_i/۲٫۵$ قراردهید. هر $\mathcal{P}(i)$ دارای ۵ ضلع است. همچنین $\mathcal{P}$ را مسیر به دست آمده از اتصال قطعه‌های $\mathcal{P}(۱), \ldots, \mathcal{P}(n)$ قراردهید. به علاوه $\mathcal{R}$ را چندضلعی حاصل از اتصال قطعات $\mathcal{R}(۱), \ldots, \mathcal{R}(n)$ قرار دهید به طوری که ابتدای آن با یک ضلع اضافی و انتهای آن با سه ضلع اضافی بسته شده‌اند. بدین ترتیب $\mathcal{P}$ دارای حداکثر $۵n$ ضلع خواهد بود. هر $\mathcal{R}(i)$ دارای ۱۲ ضلع است بنابراین $\mathcal{R}$ دارای $۱۲n + ۴$ ضلع خواهد بود. شکل ۷.۶ نمونه‌ای از این کاهش را نشان می‌دهد. قطعه $\mathcal{R}(i)$ و $\mathcal{P}(i)$ به گونه‌ای ساخته شده اند که در آن به جز نقطه‌های r_i و t_i که همدیگر را می‌بینند، سایر نقطه‌ها تنها قادر به دیدن نقطه‌های همسایه خود هستند. چندضلعی قابلیت دید r_it_i در شکل ۶.۶ با رنگ خاکستری نشان داده شده است. بنابراین تفاوت مساحت $\mathrm{WVA}(\mathcal{P}(r_i, t_i)) - \mathrm{WVA}(r_is_i)$ برابر با $۲٫۵x_i$ که همان a_i است، خواهد بود.

حال یک ساده‌سازی $\mathcal{Q}$ از $\mathcal{P}$ با اندازه‌ی k و با خطای C را در نظر بگیرید. آنگاه در زیرمسیر $\mathcal{P}(i)$ ، $\mathcal{Q}$ نمی‌تواند شامل هیچ میانبری باشد، مگر احتمالا میانبر r_it_i. روشن است که $\mathcal{Q}$ باید از میان تمام نقطه‌های p_i، $۱ \le i \le n + ۱$ بگذرد. $\mathcal{Q}_i$ را برابر با زیرمسیر $\mathcal{Q}$ از p_i به $p_{i+۱}$ قرار دهید. هر $\mathcal{Q}_i$ شامل حداقل ۴ پاره‌خط است، بنابراین $\mathcal{Q}$ حداقل $۴n$ پاره‌خط خواهد داشت. اگر $\mathcal{Q}$ دارای دقیقا $۵n$ پاره‌خط باشد، می‌توان نتیجه گرفت که هر $\mathcal{Q}_i$ دارای ۵ پاره‌خط به صورت $p_i, r_i, s_i, t_i, u_i, v_i$ بوده است. در این حالت خطای تقریب Q_i برابر صفر است و این بدان معنی است که هیچ عضوی از A انتخاب نشده است و C برابر صفر بوده است. اگر

[۱] Subset-sum



شکل ۶.۶: مشخصات قطعه‌ی iام. میانبر r_it_i در این قطعه با نقطه-خطچین نشان داده شده است. ناحیه‌ی خاکستری $\mathrm{WVP}(r_is_i)$ را نشان می‌دهد. نقطه‌های r_i و t_i به ترتیب قادر به دیدن u_i و p_i نیستند. همچنین نقطه‌ی p_{i+1} قادر به دید t_i نمی‌باشد.

مسئله پاسخ $\mathcal{Q}$ی ارایه دهد که خطای تقریب مسیر برابر C باشد آنگاه کافی است به عنوان پاسخ مسئله اصلی کلیه میانبرهای $\mathcal{Q}_i$هایی که شامل میانبر r_it_i بوده‌اند را شناسایی و a_iهای متناظر با $\mathcal{Q}_i$ را گزارش نمود.

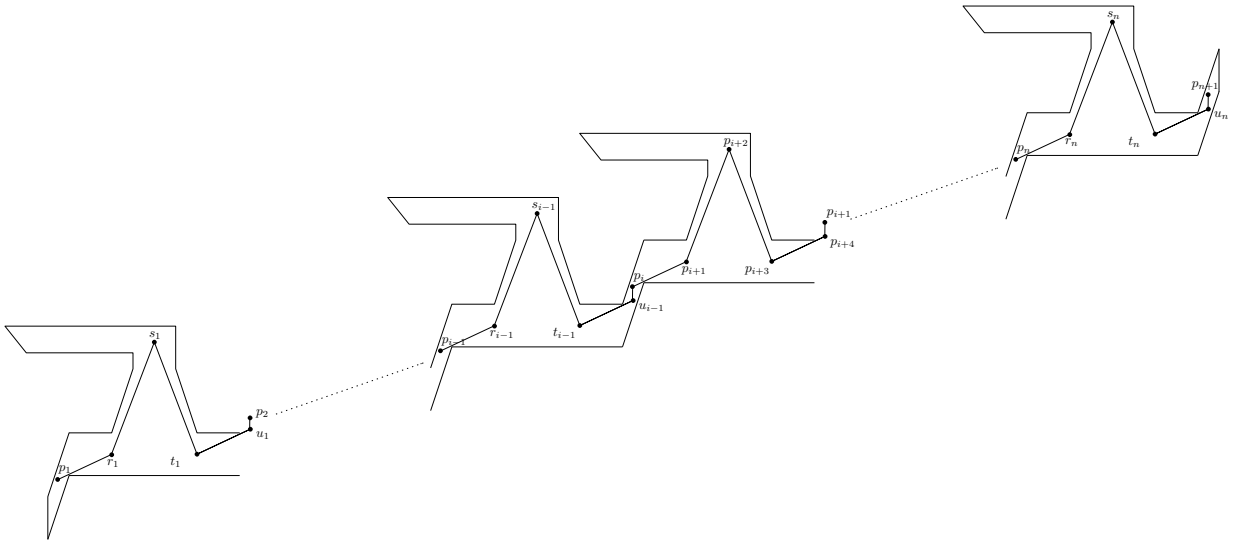
□

۲.۲.۶ الگوریتم با خطای- جمع- محدود

یک مسیر چندضلعی $\mathcal{P}$ در داخل یک چندضلعی $\mathcal{R}$، یک عدد صحیح k، یک پارامتر σ و یک پارامتر $\epsilon > 0$ به ما داده شده است. e را یک ضلع از $G_{\epsilon'}$ در نظر بگیرید، که در آن $E_\epsilon = \{p_ip_j | 1 \le i < j \le n, 0 \le \mathrm{Err}_{\mathrm{WVA}}(p_ip_{i+1}) \le \epsilon'\}$. برای سادگی قرار دهید $\mathrm{wva}(e) = \mathrm{Err}_{\mathrm{WVA}}(p_ip_j)$ و $\mathrm{WVA}(\mathcal{Q}) = \mathrm{Err}^{\mathrm{sum}}_{\mathrm{WVA}}(\mathcal{Q}) = \sum_{e\in\mathcal{Q}} \mathrm{wva}(e)$، که در آن p_ip_j میانبر مربوط به e در $\mathcal{P}$ است. هدف یافتن یک مسیر ساده تقریبی $\mathcal{Q}_{\mathrm{apx}}$ با حداکثر k ضلع است به طوری که $\mathrm{WVA}(\mathcal{Q}_{\mathrm{apx}}) \le \min_{\mathcal{Q}}\{\mathrm{WVA}(\mathcal{Q})\} + \sigma\mathcal{H}$، و در آن $\mathcal{H}$ بنابراین مقدار مساحت چندضلعی قابلیت دید $\mathcal{P}$ است. توجه داشته باشید که $\epsilon \le \mathcal{H}$. همچنین قرار دهید $\Phi = \sigma\epsilon/2k$. برای هر ضلع e از $G_{\epsilon'}$، قرار دهید $\overline{\mathrm{wva}}(e) = \lfloor \mathrm{wva}(e)/\Phi \rfloor$. با توجه به این که داریم $0 < \mathrm{wva}(e) \le \epsilon$، مقدار $\overline{\mathrm{wva}}(e)$ در محدوده $0, \ldots, \lfloor 2k/\sigma \rfloor$ است.[۲]

همچنین، برای یک مسیر $\mathcal{Q}$، قرار دهید $\overline{\mathrm{WVA}} = \sum_{e\in\mathcal{Q}} \overline{\mathrm{wva}}(e)$. همچنین $\overline{\mathrm{wva}}\Phi \le \mathrm{wva}(e) \le$

[۲] می‌توانیم از جستجوی دودویی برای محاسبه هر تابع جزء صحیح استفاده نموده و آن را در زمان $O(\log(\epsilon/\Phi))$ محاسبه نمود. بنابراین، محاسبه تابع جزء صحیح برای همه $O(n^2)$ میانبر، در زمان $O(n^2 \log k + n^2 \log(1/\sigma))$ قابل انجام است. این زمان کل زمان الگوریتم تقریبی را، که عبارت است از $O(n^2k^2/\sigma)$، تحت تاثیر قرار نمی‌دهد.



شکل ۷.۶: یک کاهش برای اثبات انپی-سخت بودن مسئله‌ی ساده‌سازی تحت معیار مجموع مساحت قابلیت دید. قطعه‌های اول و آخر به ترتیب دارای یک و دو ضلع اضافی نسبت به سایر قطعه‌ها هستند.

$(\overline{\text{wva}}+۱)\Phi$ است. بنابراین، برای ساده‌سازی $\mathcal{Q}$، داریم:

$$\begin{aligned}\overline{\text{WVA}}(\mathcal{Q})\Phi \le \text{WVA}(\mathcal{Q}) \le& \sum_{e\in\mathcal{Q}}((\overline{\text{wva}}(e)+۱)\Phi)\\ =&\overline{\text{WVA}}(\mathcal{Q})\Phi + k\Phi\\ \le&\overline{\text{WVA}}(\mathcal{Q})\Phi + \sigma\epsilon/۲\end{aligned}$$

بنابراین، داریم:

$$\text{WVA}(\mathcal{Q}) - \overline{\text{WVA}}(\mathcal{Q})\Phi \le \sigma\epsilon/۲.$$

حال، خطا را برای یک مسیر $\mathcal{Q}_{\text{apx}}$ به طوری که مقدار $\overline{\text{WVA}}(\mathcal{Q}_{\text{apx}})$ کمینه شود، محاسبه می‌کنیم. روشن است که این پاسخ مورد نظر مسئله است. پاسخ بهینه مسیر را $\mathcal{Q}_{\text{opt}}$ قرار دهید به طوری که $\text{WVA}(\mathcal{Q}_{\text{opt}})$ را کمینه کند. بنابراین، داریم:

$$\begin{aligned}\text{WVA}(\mathcal{Q}_{\text{apx}}) \le&\Phi\overline{\text{WVA}}(\mathcal{Q}_{\text{apx}}) + \sigma\epsilon/۲\\ \le&\Phi\overline{\text{WVA}}(\mathcal{Q}_{\text{opt}}) + \sigma\epsilon/۲\\ \le&\text{WVA}(\mathcal{Q}_{\text{opt}}) + \sigma\mathcal{H}\end{aligned}$$

حالا، از برنامه‌ریزی پویا و الگوریتم ارایه شده در [۱۶] استفاده می‌کنیم و به محاسبه $\mathcal{Q}_{\text{apx}}$ می‌پردازیم. $\mathcal{D}[i,r,c]$ را یک مقدار بولی قرار داده و آن را برابر با true قرار دهید اگر یک مسیر $\mathcal{Q}$ از $p_۱$ به p_i با حداکثر r ضلع وجود داشته باشد و $\overline{\text{WVA}}(\mathcal{Q}) = c$، که در آن i,r,c مقدارهای صحیح با $۱ < i \le n$، $۱ \le r \le k$ و $۰ \le c \le \lfloor ۲k/\sigma \rfloor$ هستند. ما $\mathcal{D}[i,r,c]$ را به صورت بازگشتی به طریق زیر محاسبه می‌کنیم:

- $r = ۱$:

  - if ($\overline{\text{wva}}(p_۱p_i) = c$ & $\overline{\text{wva}}(p_۱p_i) \leq \epsilon$) $\mathcal{D}[i, ۱, c] \leftarrow$true
  - else $\mathcal{D}[i, ۱, c] \leftarrow$false

- $r > ۱$:

  - if (($\mathcal{D}[i, r-۱, c]$ =true) or ($\mathcal{D}[j, r-۱, c-\overline{\text{wva}}(p_jp_i)]$ =true & $\overline{\text{wva}}(p_۱p_i) \leq \epsilon$)) $\mathcal{D}[i, ۱, c] \leftarrow$true, $۱ < j < i$

بنابراین قضیه زیر را خواهیم داشت:

قضیه ۲.۲.۶. *برای یک مسیر چندضلعی $\mathcal{P}$ داخل یک چندضلعی ساده $\mathcal{R}$، یک مقدار صحیح k، یک پارامتر ϵ، و یک پارامتر σ، یک مسیر ساده شده $\mathcal{Q}$ تحت معیار WVA با حداکثر k ضلع و با جمع خطاهای حداکثر $\sigma\mathcal{H}$ بزرگتر از پاسخ بهینه را می‌توان در زمان $O(n^۲k^۲/\sigma)$ با استفاده از $O(nk^۲/\sigma)$ فضا محاسبه نمود، که در آن $\mathcal{H}$ مساحت چندضلعی قابلیت دید $\mathcal{P}$ است.*

۳.۶ جمع‌بندی

در این بخش به مسئله ساده‌سازی مسیر محصور در یک چندضلعی ساده تحت معیارهای قابلیت دید ضعیف پرداختیم. ما یک الگوریتم چندجمله‌ای برای مسئله #-کمینه تحت معیار اندازه قابلیت دید ضعیف ارایه نمودیم. الگوریتم ما قادر است با انجام پیش پردازش بر روی چند ضلعی، به پرسمان‌های مسیر در زمان $O(n(n+m))$ پاسخ دهد. برای حل این مسئله، به مسئله شمارش اضلاع قابلیت دید ضعیف توسط یک پاره‌خط نیز پرداختیم. نشان دادیم که با انجام پیش پردازش با زمان $O(m^۷)$ و حافظه $O(m^۶)$ بر روی چندضلعی، می‌توان این مسئله را به ازای هر پاره‌خط دلخواه در زمان $O(۱)$ پاسخ داد. در ادامه ثابت نمودیم که مسئله ساده‌سازی برای جمع خطای مسیر و تحت معیار مساحت قابلیت دید ضعیف یک مسئله ان‌پی-سخت است. سپس یک الگوریتم تقریبی با جمع خطای محدود برای آن ارایه نمودیم.

فصل ۷

جمع‌بندی و کارهای آتی

در این پایان نامه به مسئله ساده‌سازی هندسی پرداختیم و الگوریتم‌هایی را برای گونه‌های مختلف این مسئله ارایه دادیم. در فصل ۴ به مسئله ساده‌سازی مسیر تحت معیارهای مساحت پرداختیم. ابتدا یک تعریف یکپارچه برای جمع- مساحت و تفاضل مساحت که قابل اعمال بر روی مسیرهای عمومی است، ارایه نمودیم. سپس، با استفاده از این تعریف، ابتدا یک الگوریتم تقریبی برای مسئله ساده‌سازی مسیر تحت معیار جمع- مساحت و برای بیشینه خطای مسیر ارایه نمودیم. مزیت این الگوریتم نسبت به روش‌های قبلی این است که اولا بر روی مسیرهای عمومی کار می‌کند. ثانیا دارای زمان اجرای نزدیک به درجه‌ی دو است. همچنین با استفاده از این تعریف به مسئله ساده‌سازی مسیر تحت معیار تفاضل- مساحت و بیشینه خطای مسیر پرداختیم. ما با استقرا اثبات کردیم که می‌توان این مسئله را در زمان درجه‌ی دو به صورت دقیق حل نمود. الگوریتم‌های موجود در این زمینه هنوز بالاتر از مرتبه خطی هستند و ارایه الگوریتم‌های تقریبی زیر درجه دو و یا نزدیک به خطی به عنوان کارهای آتی مطرح هستند.

در فصل ۵ به ارایه الگوریتم‌هایی برای ساده‌سازی هم‌فضای مسیر پرداختیم. ما چارچوبی را ارایه دادیم که با استفاده از آن می‌توان به ساده‌سازی هم‌فضا و یا قویا- هم‌فضای مسیر تحت معیار خطای دلخواه پرداخت. ابتدا این چارچوب را برای مسیرهای x- یکنوا ارایه نمودیم. الگوریتم ما برای این مسیرها بیش از $\log n$ مرتبه بهتر از الگوریتم‌های قبلی است. سپس الگوریتمی برای ساده‌سازی قویا- هم‌فضا برای مسیرهای عمومی ارایه نمودیم. در ادامه مسئله را در حالت کلی‌تر بررسی و الگوریتمی برای ساده‌سازی هم‌فضای مسیرهای عمومی ارایه نمودیم. تا آن جایی که ما می‌دانیم، دو الگوریتم آخر، اولین الگوریتم‌هایی هستند که این مسئله را برای مسیرهای عمومی به صورت بهینه حل می‌کنند. در نهایت یک الگوریتم مکاشفه‌ای ارایه دادیم که بر روی مسیرهای عمومی قابل اجرا است و زمان اجرای کمتری نسبت به الگوریتم بهینه ما و الگوریتم‌های مکاشفه‌ای قبلی دارد. برای کارهای آتی در این زمینه می‌توان بر روی بهبود زمان اجرای الگوریتم بهینه برای مسیرهای عمومی کار نمود. کار دیگری

که به عنوان کارهای آتی پیشنهاد می‌شود، مطالعه مسئله ساده‌سازی هم‌فضا تحت یک معیار خطای خاص مانند هاسدورف یا فرشه است. ممکن است بتوان با در نظر گرفتن خصوصیات یک معیار خطای خاص، الگوریتمی با زمان اجرای بهتر ارایه داد.

در فصل ۶ الگوریتم‌هایی را برای ساده‌سازی تحت معیار قابلیت دید ضعیف ارایه نمودیم. ابتدا به مسئله ساده‌سازی تحت معیار خطای تعداد ضلع‌های قابلیت دید ضعیف و بیشینه خطای مسیر پرداختیم. برای این مسئله الگوریتمی ارایه نمودیم که با انجام پیش پردازش بر روی چندضلعی قادر است، برای مسیر دلخواه، مسیر بهینه را در $O(n)$ مرتبه سریعتر نسبت به الگوریتم ابتدایی حل نماید. سپس به مسئله ساده‌سازی مسیر تحت معیار خطای مساحت قابلیت دید ضعیف پرداختیم. ما نشان دادیم که این مسئله برای مجموع خطای مسیر، یک مسئله ان‌پی-سخت است. در نهایت یک الگوریتم تقریبی برای آن ارایه نمودیم. به عنوان کارهای آتی می‌توان بر روی بهبود زمان پیش پردازش الگوریتم تحت معیار تعداد ضلع‌های قابلیت دید ضعیف کار نمود.

پیوست آ

الگوریتم ساده‌سازی زمین با استفاده از پردازنده‌ی گرافیکی

در این بخش به مساله ساده‌سازی زمین (گستره جغرافیایی) می‌پردازیم. این مساله حالت دو و نیم بعدی مساله ساده‌سازی هندسی است. در این مساله، زمین به صورت مجموعه‌ای از نقطه‌ها بر روی صفحه به ما داده می‌شود. هر نقطه علاوه بر مختصات x و y دارای یک ارتفاع نیز هست. هدف یافتن مجموعه‌ای از نقطه‌ها با تعداد کمتر است به طوری که شکل کلی زمین داده شده حفظ شود. ساده‌سازی زمین از آن جایی اهمیت دارد که فرآیند ساده‌سازی سبب می‌شود تا تعداد نقطه‌های ورودی بسیار کاهش یابد. در نتیجه فضای لازم برای نگهداری این داده‌ها و همچنین زمان لازم برای انجام پردازش بر روی آنها بسیار بهبود یابد. در این بخش ما الگوریتم موازی مکاشفه‌ای برای ساده‌سازی زمین مبتنی بر پردازنده گرافیکی ارایه می‌دهیم. این الگوریتم خطای زمین ساده‌شده را نسبت به زمین ورودی تضمین می‌کند. این موضوع سبب می‌شود تا خطای انجام محاسبه‌ها بر روی چنین داده‌ای مشخص باشد. الگوریتمی که ما پیشنهاد می‌دهیم یک الگوریتم پیوندی سی‌پی‌یو-جی‌پی‌یو است. این الگوریتم حجم انتقال داده بین سی‌پی‌یو و جی‌پی‌یو را کاهش می‌دهد. ما روش پیشنهادی را بر روی مجموعه داده‌های مختلف با اندازه‌های متفاوت و در نظر گرفتن پارامترهای خطای متفاوت اجرا می‌نماییم. برای داده‌هایی که در حافظه جی‌پی‌یو جای می‌گیرند، ما به طور متوسط به تسریع ۱۳x نسبت به الگوریتم مشابه بر روی سی‌پی‌یو دست یافتیم. برای داده‌های بزرگ که در این حافظه جای نمی‌گیرند، و نیاز به چندین بار انتقال داده به جی‌پی‌یو است، ما به ۴x افزایش سرعت دست یافتیم. روش ارایه شده به ژورنال زیر ارسال شده است:

1. S. Daneshpajouh, M. Ghodsi, M. Gholami, S. Keshtkar Jafari, M. Mahmoudi Aznaveh, TIN Simplification Algorithm on Graphical Processing Units, Submitted to Computers & Geosciences, 2013.

آ.۱ انگیزه پژوهش در زمینه‌ی ساده‌سازی سطح‌ها

در بسیاری از کاربردها نمایش شیءها به صورت یک مسیر یا منحنی مناسب نیست و شیء به صورت یک سطح چندضلعی نمایش می‌یابد. در این گونه کاربردها، شیء را با چندضلعی پیچیده‌تری مانند شبکه‌های نامنظم مثلث‌بندی شده[۱] یا به اختصار تین [۲] نمایش می‌دهند. در روش‌های ساده‌سازی سطح‌ها، هدف به دست آوردن یک سطح ساده‌تر از ورودی داده شده است، به طوری که سطح به دست آمده تقریب مناسبی از سطح اصلی باشد. از جمله کاربردهای این مسئله می‌توان به ساده‌سازی یک ناحیه یا گستره جغرافیایی اشاره نمود. مانند ساده‌سازی منحنی، در این کاربردها نیز میزان نگهداری جزئیات مطرح است و هدف نگهداری الگوی کلی شکل اصلی ولی با پیچیدگی کمتر است.

یکی از مهمترین کاربردهای این مسئله در بینایی ماشین است. به عنوان مثال پویش‌گرهای لیزری میزان زیادی از داده‌ها را ضبط می‌کنند. در کنترل از راه دور، داده‌های زمین توسط ماهواره‌های تصویربرداری دریافت می‌شود و در گرافیک کامپیوتری و طراحی هندسی به کمک کامپیوتر، مدل‌های چندضلعی توسط مجموعه‌ای از نواحی از سطح‌های منحنی با پارامترهای معین، تولید می‌شوند. در همه این مثال‌ها شیءها به صورت مدل‌هایی از سطح‌ها و شامل میلیون‌ها چندضلعی در می‌آیند. در این مسایل، ساده‌سازی به این دلیل اهمیت دارد که استفاده از فضا، سرعت انتقال داده، زمان انجام محاسبات و نمایش شکل‌ها با کارایی بیشتر انجام پذیرد.

کاربرد دیگر این مسئله در گرافیک کامپیوتری و زمینه‌های مرتبط با آن مانند واقعیت مجازی[۳]، طراحی هندسی به کمک کامپیوتر، تصویرسازی علمی، فشرده‌سازی داده‌ها و نمایش سریع شکل‌ها است. به علاوه در برنامه‌های تعاملی[۴] مانند برنامه‌های شبیه‌ساز پرواز، بازی‌های کامپیوتری و طراحی کامپیوتری، کارایی و برخط بودن برنامه بسیار مهم است. برای چنین کاربردهایی، شکل هندسی می‌تواند در سطوح مختلف دارای میزان دقت متفاوتی باشد. به عبارت دیگر میزان دقت نمایش شکل شیء می‌تواند باتوجه به میزان نزدیکی یا دور بودن تنظیم شود. این تکنیک تحت عنوان مدل‌سازی با قدرت تفکیک متعدد[۵] شناخته شده است. برای حالت غیر-برخط نیازمندی‌های وضعیت برخط حیاتی نیست ولی سرعت و حافظه مناسب مهم است. از دیگر کاربردهای این مسئله می‌توان به انتقال و ارسال شیءها سه‌بعدی اشاره نمود. در این کاربرد فشرده‌سازی اهمیت بسیاری پیدا می‌کند. با توجه به حجم بالای داده چه پهنای باند کانال ارتباطی کم باشد (مانند مودم) و چه پهنای باند زیاد باشد (مانند خطوط اصلی اینترنت) فشرده‌سازی دارای اهمیت است. بنابراین ساده‌سازی می‌تواند به عنوان یک

[۱] Triangulated Irregualr Network

[۲] TIN

[۳] Virtual Reality

[۴] Interactive

[۵] Multi Resolution

راه‌حل در این کاربردها کمک کند.

آ.۲ کودا

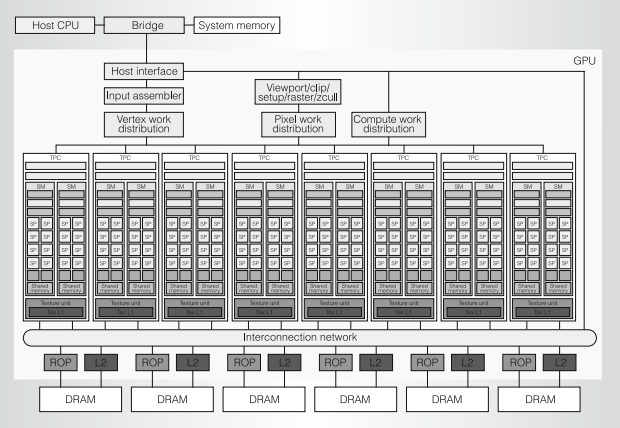
در سال ۲۰۰۷ انویدیا[۶] یک معماری محاسباتی و مدل برنامه نویسی جدید با نام کودا[۷] ارایه نمود. کودا امکانات جدیدی را برای برنامه‌سازی موازی بر روی جی‌پی‌یوها به برنامه‌نویس‌ها می‌دهد. در واقع این چارچوب با پنهان‌کردن تابع‌های سطح پایین گرافیکی، برنامه‌نویسی موازی بر روی جی‌پی‌یوها را تسهیل می‌بخشد.علت گسترش روزافزون استفاده از چنین پردازش‌گرهایی سرعت بیشتر پیشرفت این‌گونه پردازش‌گرها نسبت به سی‌پی‌یوهای معمولی است. با توجه به این که این گونه پردازش‌گرها نه‌تنها در کامپیوترهای شخصی بلکه در وسایل سرگرمی (مانند PlayStation ۳ و Xbox) نیز کاربرد دارند، بازار گسترده‌تری نیز پیدا کرده‌اند. این بازار باعث رقابتی‌تر شدن فضای تولید و در نتیجه رشد سریع پردازنده‌های گرافیکی شده است.

در هر جی‌پی‌یو تعدادی Streaming Multiprocessor یا به اختصار اس‌ام وجود دارد که هرکدام یک واحد کنترلی، حافظه‌ی نهان و تعدادی ALU دارند. تعداد اس‌ام‌ها در هر جی‌پی‌یو و تعداد ALUها در هر اس‌ام، بسته به نوع کارت گرافیکی متفاوت است. شکل آ.۱ معماری کارت گرافیک GeForce ۸۸۰۰ را نشان می‌دهد که در آن هشت اس‌ام و در هر اس‌ام شانزده واحد پردازشی وجود دارد. این واحدها از طریق شبکه‌ی اتصالی به واحد حافظه‌ی کارت گرافیکی متصل هستند. در هر اس‌ام تعدادی رجیستر با سرعت بالا و حافظه‌های پنهان نیز وجود دارد که برنامه‌نویس می‌تواند از آن‌ها استفاده کند. با توجه به کمبود حافظه‌ی نهان جی‌پی‌یو نسبت به سی‌پی‌یو مدیریت مناسب حافظه‌ها توسط برنامه‌نویس اهمیت زیادی در افزایش سرعت دارد.

ار آنجایی که هر اس‌ام تنها یک واحد کنترلی دارد تمام این واحدها یکسان عمل می‌کنند به عبارت دیگر هر اس‌ام، به شکل Single Instruction Multiple Data یا به اختصار SIMD عمل کرده و داده‌ها را پردازش می‌کند. برنامه‌هایی برای موازی‌سازی بر روی جی‌پی‌یو مناسب‌تر هستند که از یک طرف میزان موازی‌سازی زیادی داشته باشند و از طرف دیگر نیز در مدل SIMD به خوبی بگنجند. در این معماری، هر نخ به رجیسترهای خود، حافظه‌ی مشترک و حافظه‌ی سراسری دسترسی دارد. رجیسترها سریع‌ترین حافظه‌ها هستند، اما تعداد محدودی از آن‌ها در اختیار است. حافظه مشترک بین نخ‌های درون یک اس‌ام، از نظر سرعت در جایگاه بعدی قرار می‌گیرند. میزان این حافظه محدود است و بیشتر برای هماهنگی بین نخ‌های یک اس‌ام استفاده می‌شوند. حافظه‌ی سراسری کندترین و در عین حال حجیم‌ترین حافظه در جی‌پی‌یو می‌باشد. هرچند این حافظه در برخی

[۶] NVIDIA

[۷] CUDA



شکل آ.۱: معماری کودا[۴۸].

از مدل‌ها حافظه‌ی نهان دارد، اما همچنان استفاده از آن می‌تواند سرعت برنامه را کاهش دهد. راه حلی که برنامه‌نویسان برای غلبه بر این مشکل به کار می‌برند، استفاده از تعداد زیادی نخ است. به این ترتیب زمانی که تعدادی از نخ‌ها منتظر آماده شدن داده‌های مورد نیازشان هستند، نخ‌های دیگری که داده‌های آنها آماده است اجرا می‌شوند. به این ترتیب زمان فراخوانی داده با تعداد زیاد نخ‌ها پوشیده می‌شود. برای گرفتن بهترین نتیجه در جی‌پی‌یو علاوه بر ایجاد تعداد بسیار زیادی نخ، نخ‌هایی که با هم اجرا می‌شوند نباید میزان زیادی انشعاب داشته باشند. تعداد نخ‌هایی که بر روی هر اس‌ام اجرا می‌شوند، با توجه به نوع کارت گرافیکی متفاوت است. برنامه‌نویس با توجه به نوع کارت گرافیکی باید این تعداد را مدیریت کند. در صورتی که در اجرای تعدادی نخ در یک اس‌ام یک انشعاب وجود داشته باشد، نخ‌ها دو دسته می‌شوند و نخ‌هایی که در یک دسته قرار دارند با هم اجرا می‌شوند و نخ‌های دیگر منتظر می‌مانند. در حقیقت در جی‌پی‌یو منابع اولیه از جمله رجیسترها که به نخ اختصاص داده شد تا پایان کار مختص همان نخ باقی می‌مانند. هرچند این عمل از لحاظ منابع محدودیت به وجود می‌آورد اما باعث می‌شود که تعویض نخ‌های بر روی هر اس‌ام با سرعت زیادی انجام پذیرد. از همین رو نخ‌های روی جی‌پی‌یو را بی‌وزن می‌نامند.

آ.۳ کارهای مرتبط

تاکنون، الگوریتم‌های تقریبی و مکاشفه‌ای بسیاری برای مسئله ساده‌سازی زمین یا تین ارایه شده است. آگاروال و سوری [۴۹] یک الگوریتم تقریبی برای ساده‌سازی زمین ارایه دادند. الگوریتم آنها در زمان $O(n^{۸})$ و با اندازه $O(k \log k)$ اجرا می‌شود که در آن k اندازه بهینه ϵ-تقریب است. بعد از آن آگاروال و دسیکان [۵۰] الگوریتم تقریبی دیگری برای این مسئله پیشنهاد کردند که دارای زمان $O(n^{۲+\delta} + c^{۳} \log^{۲} c \log \frac{n}{c})$ (متوسط زمان انتظار) و با اندازه خروجی $O(c^{۲} \log^{۲} c)$ است که در آن c اندازه ϵ-تقریب با کمینه تعداد نقطه‌ها است.

اگر این الگوریتم دارای پیچیدگی زمانی بهتری از نظر نظری است، اما (تا آن جایی که ما می‌دانیم) به دلیل برخی پیچیدگی‌های پیاده‌سازی چندان در برنامه‌های کاربردی مورد استفاده قرار نگرفته است.

الگوریتم‌های مکاشفه‌ای ساده‌سازی تین را می‌توان به دو شاخه تقسیم نمود؛ پالایش[۸] و [۹]حذف. در الگوریتم‌های مبتنی بر پالایش، ابتدا یک تین خروجی که از دو مثلث تشکیل شده است، ساخته می‌شود. سپس نقطه با بیشترین خطا در هر مثلث جستجو شده و به مثلث مربوطه اضافه می‌شوند. بعد از اضافه کردن این نقطه‌ها، تین خروجی دوباره مثلث‌بندی می‌شود. الگوریتم این مرحله‌ها را تکرار می‌کند تا زمانی که هیچ نقطه‌ای با خطای بزرگتر از خطای داده شده‌ی ϵ، در کلیه مثلث‌ها، وجود نداشته باشد [۵۱].

الگوریتم‌های مبتنی بر حذف، بر روی خود تین اصلی ساده‌سازی را انجام می‌دهند. در هر مرحله از الگوریتم، یک نقطه با خطای کمینه از تین حذف می‌شوند. سپس، تین مجددا مثلث‌بندی می‌شود. الگوریتم این مرحله‌ها را تکرار می‌کند تا آن که هیچ نقطه‌ای با خطای بزرگتر از خطای داده شده ϵ باقی نمانده باشد، یا آن که اندازه تین ساده شده برابر با اندازه m خواسته شده (با توجه به هدف تعریف شده) شود [۵۲].

گارلند و دیگران [۳] روش حذف را با ایده فشرده‌کردن لبه، ترکیب کردند. بنابراین، یک یا چند لبه کم اهمیت در هر مرحله از الگوریتم فشرده می‌شوند. یون و همکارانش از روش حذف مبتنی برای حذف یک یا چند نقطه با اهمیت کم‌تر در هر مرحله از الگوریتم، استفاده نمودند.

اخیرا، تعدادی الگوریتم مبتنی بر جی‌پی‌یو برای ساده‌سازی زمین ارایه شده‌اند [۸]. این روش‌ها بیش‌تر برای رندرکردن سریع یک زمین طراحی شده‌اند و برای کاربردهای گرافیکی و برخط مناسب هستند. متاسفانه، اکثر این روش‌ها تضمین نمی‌کنند که فاصله بین زمین ساده شده تا زمین اصلی بیش‌تر از اندازه‌ی ϵ نباشد. علت این است که در این کاربردها فاصله کاربر (یا دوربین) از زمین به عنوان معیار خطا مد نظر قرار می‌گیرد. به این ترتیب شیءهای نزدیکتر با کیفیت بهتر و شیءهای دورتر با کیفیت کمتری نمایش می‌یابند. بیشتر این روش‌ها، زمین را به قسمت‌های لوزی شکل تقسیم نموده و به صورت سلسه مراتبی و با توجه به محل دوربین به ساده‌سازی ناحیه می‌پردازند. در بسیاری از کاربردهای محاسباتی (مانند پیش‌بینی سیل، افزایش سطح دریاها، و ...)، میزان خطای تین ساده‌شده نسبت به تین اصلی اهمیت دارد و معمولا دقت یکسان در کل زمین مد نظر است. بنابراین، الگوریتم سریعی که یک ساده‌سازی از زمین با فاصله‌ی خطای مورد نظر ارایه دهد، مورد نیاز می‌باشد.

---

[۸] Refinement

[۹] Decimation

۱.۳.آ الگوریتم هلر

الگوریتم هلر [۵۱]، یکی از الگوریتم‌های حریصانه‌ای است که، به دلیل کنترل خطای تین ساده شده، بسیار در کاربردهای مختلف مورد استفاده قرار گرفته است. در بخش ۴.آ یک الگوریتم موازی برای مسئله ساده‌سازی تین ارایه می‌دهیم که خطای تین خروجی را تضمین می‌کند. این الگوریتم برای کنترل خطای ساده‌سازی از ایده‌های الگوریتم هلر استفاده می‌کند. در ادامه به توضیح الگوریتم هلر می‌پردازیم. این الگوریتم یک توری را به عنوان ورودی دریافت و سپس یک خروجی با تعداد کم‌تری نقطه به صورت شبکه‌ی نامنظم مثلثی (تین) ساخته و بر روی آن کار می‌کند. این الگوریتم با اضافه کردن نقطه‌ها از توری ورودی به تین خروجی، کار می‌کند و تا زمانی که نقطه‌ی دیگری با خطای بزرگتر از ϵ باقی نمانده باشد، به کار خود ادامه می‌دهد. در ادامه جزئیات این الگوریتم را ملاحظه می‌کنید.

۲-آ. الگوریتم هلر

ورودی:

$\mathcal{M}$ – توری ورودی

ϵ – میزان خطای ساده‌سازی

---

۱. مجموعه V را مجموعه‌ی نقطه‌های توری $\mathcal{M}$ قرار بده. چهار نقطه‌ی چهار طرف توری را حذف کن و آنها را به مجموعه‌ی V' اضافه کن. این مجموعه که آن را T' می‌نامیم، حاوی نقطه‌های تین ساده شده است.

۲. مثلث‌بندی دلونی $\mathrm{DT}(V')$ را محاسبه کن.

۳. برای هر نقطه در V، مثلث $\Delta_i \in \mathrm{DT}(V')$ که حاوی آن است را پیدا کن.

۴. فاصله‌ی (خطای) همه‌ی نقطه‌ها تا مثلث مربوطه را محاسبه کن.

۵. برای همه نقطه‌های $v_i \in V$، اگر $\mathrm{Err}(v_i)$ در T'، کمتر از ϵ باشد، آنگاه تین T' پذیرفته می‌شود. در غیر این صورت نقطه با بیشترین خطا را از V حذف نموده و به V' اضافه کن. سپس به مرحله ۲ برو.

۴.آ الگوریتم موازی ساده‌سازی مبتنی بر سی‌پی‌یو- جی‌پی‌یو

در این بخش ما یک الگوریتم سی‌پی‌یو- جی‌پی‌یو برای ساده‌سازی زمین، ارایه می‌کنیم. این الگوریتم مشابه روش هلر [۵۱] (در بخش ۱.۳.آ این روش به طور خلاصه توضیح داده شده است) برای ساده‌سازی زمین است. ولی

به طور خاص برای استفاده از امکانات موازی‌سازی جی‌پی‌یو توسعه یافته است. ایده کلی این است که به جای ساده‌سازی یک‌پارچه‌ی تین ورودی، آن را به تعدادی نوار تقسیم نموده و هر نوار را در اس‌ام‌های جی‌پی‌یو ساده کنیم. این ساده‌سازی دو سطح از موازی‌سازی را شامل می‌شود. سطح اول شامل ساده‌سازی نوارها به صورت موازی توسط اس‌ام‌های مختلف است. سطح دوم ساده‌سازی یک نوار به صورت موازی در داخل یک اس‌ام با بکارگیری هسته‌های آن است. به هنگام ساده‌سازی تین، ما خطا را کنترل نموده و یک مثلث‌بندی دلونی در داخل هر اس‌ام می‌سازیم. پس از این مرحله، نوارهای ساده شده به سی‌پی‌یو برای ادغام نهایی ارسال می‌شوند. در سی‌پی‌یو ، نوارها ادغام شده و مجدد ساده می‌شوند به طوری که نه تنها تین خروجی یک مثلث‌بندی دلونی باشد، بلکه فاصله آن نسبت به تین اصلی (فاصله‌ی عمودی هر نقطه تا مثلث مربوطه در تین ساده شده)، کمتر از ϵ باشد. در ادامه جزئیات روش پیشنهادی ارایه می‌شود.

۱.۴.آ الگوریتم اصلی

ابتدا در سی‌پی‌یو مجموعه نقطه‌های ورودی $V \in \mathcal{M}$ خوانده شده و آنها را برحسب مختصات xشان مرتب می‌کنیم. سپس ورودی به k ناحیه‌ی همسایه، به صورت k نوار عمودی تقسیم می‌شود. سپس فضای حافظه موردنیاز در جی‌پی‌یو اختصاص می‌یابد. این فضا شامل فهرستی از نقطه‌ها، مثلث‌ها و پاره‌خط‌ها می‌باشند. این داده‌ها در فضای حافظه‌ی سراسری جی‌پی‌یو نگهداری می‌شوند. در هربار انتقال اطلاعات از سی‌پی‌یو به جی‌پی‌یو، به تعداد اس‌ام‌ها به انتقال ناحیه‌ها می‌پردازیم. از آن جایی که در انتقال اطلاعات به جی‌پی‌یو و اجرای برنامه، نباید اندازه‌ی داده از اندازه‌ی حافظه‌ی سراسری جی‌پی‌یو تجاوز کند. بنابراین، اندازه‌ی k را با توجه به حافظه‌ی سخت‌افزار در دسترس و تعداد اس‌ام‌های سخت‌افزار تعیین می‌کنیم.

۳-آ. الگوریتم اصلی(در سی‌پی‌یو)

ورودی:

$\mathcal{M}$ – توری ورودی

ϵ – میزان خطای ساده‌سازی

---

۱. نقطه‌های توری ورودی را خوانده و آنها را بر حسب مختصات xشان مرتب کن.

۲. نقطه‌های مرتب شده را به k نوار عمودی(نسبت به محور xها) با اندازه مساوی برحسب تعداد نقطه‌ها، تقسیم کن.

۳. حداکثر به تعداد اس‌ام‌ها، نوارهای داده را به‌همراه پارامتر خطای ϵ، به جی‌پی‌یو ارسال کن.

۴. هسته‌ی جی‌پی‌یو برای اجرای *الگوریتم ساده‌سازی در اس‌ام‌ها* (الگوریتم ۴-آ) را فراخوانی و اجرا کن.

۵. پس از اتمام کار جی‌پی‌یو، نتیجه‌های ساده‌سازی‌ها را از جی‌پی‌یو دریافت و ذخیره کن.

۶. درصورتی که هنوز نواری باقی مانده باشد که پردازش نشده باشد، برو به مرحله‌ی ۳.

۷. الگوریتم *ادغام دلونی نوارها* (الگوریتم ۶-آ) در سی‌پی‌یو را فراخوانی کن.

۸. پایان الگوریتم.

۲.۴.آ الگوریتم ساده‌سازی در اس‌ام

در جی‌پی‌یو، هر اس‌ام به طور مستقل از سایر اس‌ام‌ها به ساده‌سازی نوار داده‌ی مربوط به خود می‌پردازد. V_j را مجموعه‌ی نقطه‌های ارسال شده به اس‌ام j-ام در نظر بگیرید. N_j را اندازه‌ی V_j در نظر بگیرید. همچنین v_i^j را نقطه‌ای از نقطه‌های V_j در نظر بگیرید. T'_j را مجموعه مثلث‌های دلونی ساخته شده بر روی نقطه‌های V_j توسط اس‌ام jام قرار بده. به علاوه P_j را تعداد نخ‌های فیزیکی در اس‌ام jام در نظر بگیرید. در ادامه الگوریتم مربوط به ساده‌سازی توسط یک اس‌ام را ملاحظه می‌نمایید.

۴-آ. الگوریتم ساده‌سازی در اس‌ام

ورودی:

V_j – مجموعه‌ی نقطه‌ها

ϵ – میزان خطای ساده‌سازی

۱. چهار نقطه‌ی منتهی الیه چهار سمتِ V_j را حذف و به T'_j اضافه کن.

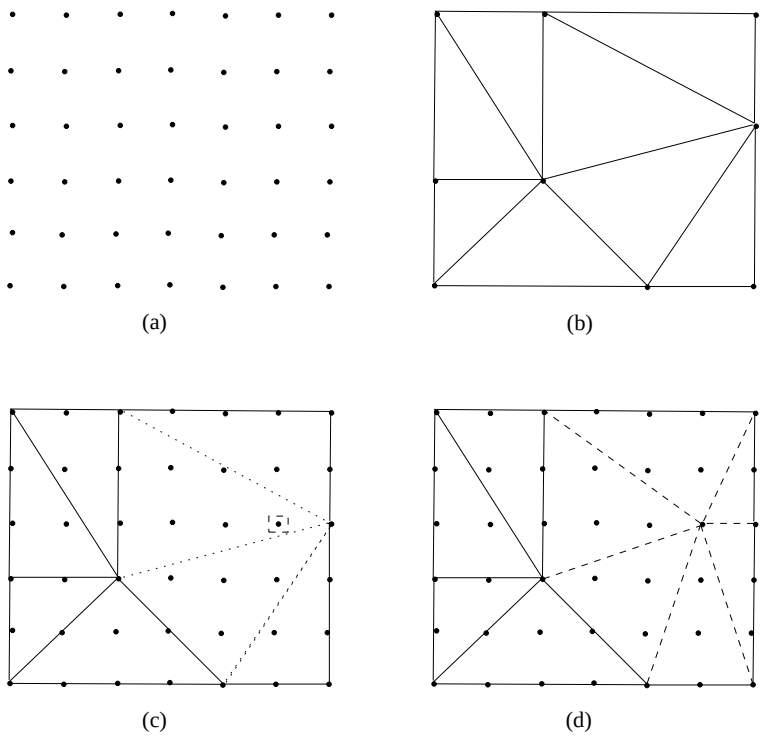
۲. خطای همه‌ی نقطه‌های V_j را محاسبه کن.

۳. نقطه $v_i^j \in V_j$ با بیشترین خطا را پیدا کن.

۴. اگر خطای v_i^j کمتر از ϵ باشد آنگاه برو به مرحله، ۵.

۵. در غیر اینصورت ، الگوریتم ۵-آ را برای اضافه‌کردن v_i^j به مثلث‌بندی دلونی T'_j فراخوانی کن و برو به مرحله‌ی ۳.

۶. پایان الگوریتم.

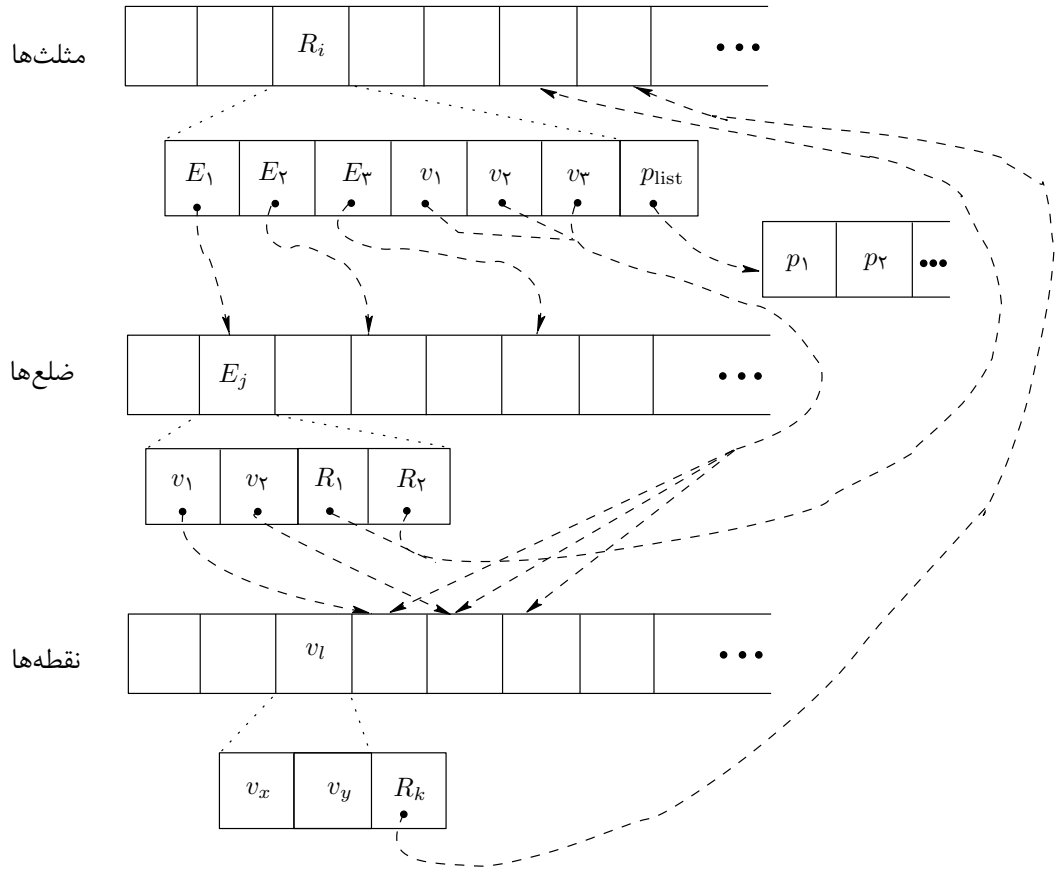


شکل آ.۲: ساده‌سازی در اس‌ام . (a) توری ورودی. (b) تین خروجی در مرحله‌ی ۱ − iام. (c) اضافه کردن نقطه با بیشترین خطا در مرحله‌ی iام و حذف ضلع‌های ناقض مثلث‌بندی دلونی(ضلع‌های نقطه چین) (d) مثلث‌بندی دلونی جدید (ضلع‌های منقطع).

در مرحله‌ی ۲ از الگوریتم ساده‌سازی در اس‌ام، خطای کلیه نقطه‌های v^j حساب می‌شوند. این کار به صورت موازی و با ایجاد نخ‌ها به تعداد نقطه‌ها انجام می‌شوند. به عبارت دیگر هر نخ به طور مستقل خطای یک نقطه را محاسبه و ذخیره می‌کند. هر مثلث یک صفحه را تعریف می‌کند. خطای یک نقطه v_i^j ، $(\mathrm{Err}(v_i^j))$ کمترین فاصله‌ی عمودی بین v_i^j و سطح صفحه است. این خطا معمولا توسط رابطه $|z_i' - z_i|$ حساب می‌شود که در آن z_i اندازه ارتفاع واقعی نقطه‌ی v_i^j بوده و z_i' ارتفاع از تصویر نقطه بر روی صفحه (نقطه با همان x و y) است. این کار در زمان $O(N/P)$ انجام می‌شود. سپس و در مرحله‌ی ۳ الگوریتم، نقطه با بیشترین خطا را به صورت موازی پیدا می‌کند. این‌کار در زمان $O(N_j/P + \log(N_j))$ قابل انجام است. در مرحله ۴ بررسی می‌شود آیا خطای نقطه با بیشترین خطا، از خطای ورودی ϵ بیشتر است یا کمتر. اگر بیشتر باشد، این نقطه با فراخوانی الگوریتم ۵-آ به مثلث‌بندی دلونی T_j' اضافه می‌شود و اگر کمتر باشد الگوریتم اس‌ام به پایان می‌رسد.

۳.۴.آ طرح داده

داده هر نوار در سه فهرست ذخیره می‌شوند: فهرستی از مثلث‌ها، فهرستی از ضلع‌ها و یک فهرست ازنقطه‌ها. در فهرست مثلث‌ها، هر سلول R_i اشاره‌گرهایی به سه ضلع مثلث‌اش، نقطه‌های رأسی‌اش و یک اشاره‌گر p_{list} به فهرست نقطه‌هایی که در داخل R_i جای می‌گیرند. در فهرست ضلع‌ها، هر ضلع E_j اشاره‌گرهایی به دوسرش



شکل آ.۳: داده‌ساختار در جی‌پی‌یو

دارد و همچنین به دو مثلث سمت چپ و راست ضلع. در فهرست نقطه‌ها، هر نقطه v_l مختصات x و y خود را دارد. شکل آ.۳ طرح داده برای یک نوار را نشان می‌دهد.

۴.۴.آ مثلث‌بندی دلونی به صورت افزایشی در یک اس‌ام

ساخت مثلث‌بندی دلونی از چندین مرحله تشکیل شده است. ما از مثلث‌بندی دلونی افزایشی استفاده می‌کنیم. R_j را مجموعه مثلث‌هایی که یک نقطه v_i درون دایره محیطی آنها قرار می گیرد، تعریف می‌کنیم. مرحله‌های ساخت مثلث‌بندی دلونی به صورت زیر می‌باشد.

۵-آ. الگوریتم اضافه‌کردن v_i به مثلث‌بندی دلونی در اس‌ام

ورودی:

V_j – مجموعه‌ی نقطه‌ها

T'_j – مجموعه‌ی مثلث‌ها

ϵ – میزان خطای ساده‌سازی

$v^j_{\max}$ – نقطه با بیشترین خطا

---

۱. پیدا کردن مجموعه‌ای از مثلث‌های R_i که نقطه‌ی $v^j_{\max}$ داخل دایره‌ی محیطی آنها قرار می‌گیرد.

۲. تخریب همه مثلث‌های R_i در T'_j، توسط حذف لبه‌های مشترک آنها.

۳. ایجاد مثلث‌های جدید دلونی R'_i در T'_j، توسط ایجاد لبه‌های جدید.

۴. مکان‌یابی نقطه‌های واقع در مثلث‌های قدیمی R_i در مثلث‌های جدید R'_i.

۵. بروزرسانی خطای نقطه‌های واقع در مثلث‌های جدید R'_i.

در این الگوریتم، نقطه با بیشترین خطا به مثلث‌بندی دلونی T'_j اضافه شده به گونه‌ای که مثلث‌بندی حاصل همچنان دلونی باشد. مرحله‌های ۱، ۴ و ۵ به صورت موازی انجام می‌شوند و به ازای هر نقطه یک نخ ایجاد می‌شود. بنابراین کل این مرحله‌ها در $O(N_j/P_j)$ انجام می‌شوند. ولی مرحله‌های ۲ و ۳ به صورت ترتیبی و تنها توسط یک نخ انجام می‌شود. شکل ۲.آ مرحله‌های انجام الگوریتم اس‌ام را نشان می‌دهد.

۵.۴.آ مرحله ادغام در سی‌پی‌یو

حال که توضیح مجموعه کارهایی که در جی‌پی‌یو انجام می‌شوند به پایان رسید به توضیح نحوه ادغام نوارهای ساده شده توسط اس‌ام‌ها می‌پردازیم. همان‌طور که گفته شده این کار در سی‌پی‌یوانجام می‌شود. جزئیات آن در الگوریتم ۶-آ آمده است. در این الگوریتم ما ابتدا خروجی ناحیه‌های اس‌ام‌ها را ادغام می‌کنیم (خط‌های ۱ الی ۲). سپس مثلث‌بندی دلونی را بروز رسانی کرده و خطاهای نقطه‌ها در مثلث‌های جدید بروز شده را محاسبه می‌کنیم (خط‌های ۳ الی ۸). در نهایت، ما نقطه‌های با خطای بزرگتر از ϵ را پیدا کرده و مثلث‌بندی دلونی را بروز می‌کنیم. این کار را تا زمانی که نقطه‌ای با خطای بزرگتر از ϵ نباشد ادامه می‌دهیم (خط‌های ۱۰ الی ۱۳). بنابراین، خروجی تین ما حتما یک مثلث‌بندی دلونی خواهد بود در عین حال که خطای همه نقطه‌ها در تین اصلی بزرگتر از ϵ نخواهد بود.

۶-آ. الگوریتم ادغام دلونی نوارها در سی‌پی‌یو

ورودی:

$V_j, j \in \{۱, ۲, \ldots, k\}$ – مجموعه‌ی نقطه‌ها

$T'_j, j \in \{۱, ۲, \ldots, k\}$ – مجموعه‌ی مثلث‌ها

$S'_j, j \in \{۱, ۲, \ldots, k\}$ – مجموعه‌ی پاره‌خط‌ها

ϵ – میزان خطای ساده‌سازی

---

۱. اتصال k نوارهای(ناحیه‌های) همسایه با یکپارچه نمودن مثلث‌های مرزی آن‌ها.

۲. اضافه‌کردن مثلث‌های مرزی به فهرست مثلث‌های در دست بررسی ListofTrianglesForCheck.

۳. برای همه‌ی مثلث‌های $R_i \in$ ListofTrianglesForCheck بررسی کن آیا یک مثلث‌بندی دلونی است یا خیر.

۴. اگر همه‌ی مثلث‌ها دلونی بودند برو به مرحله‌ی ۱۰.

۵. برای هر مثلث R_i، که مثلث‌بندی دلونی نیست، آن را به همراه مثلث‌های همسایه آن و پاره‌خط‌های مربوطه حذف کن. همچنین آن را از ListofTrianglesForCheck و ListofUpdatedTriangles (اگر در آن باشد) حذف کن.

۶. ایجاد مثلث‌های جدید دلونی R'_i در T'، توسط ایجاد لبه‌های جدید و اضافه‌کردن همسایه‌های این مثلث‌ها به ListofTrianglesForCheck و ListofUpdatedTriangles.

۷. مکان‌یابی نقطه‌های واقع در مثلث‌های قدیمی R_i در مثلث‌های جدید R'_i.

۸. بروزرسانی خطای نقطه‌های واقع در مثلث‌های جدید R'_i.

۹. برو به مرحله‌ی ۳.

۱۰. برای همه نقطه‌های $v_i \in V \in$ *ListOfUpdatedTriangles*، اگر $Error(v_i)$ کمتر از ϵ باشد، برو به مرحله ۱۳. در غیر اینصورت، نقطه با خطای بیشینه را از V حذف کن و آن را به مثلث‌بندی دلونی اضافه نموده و عمل بروزرسانی را انجام بده.

۱۱. V را از *ListOfUpdatedTriangles* حذف کن و مثلث جدید را به *ListOfUpdatedTriangles* اضافه کن.

۱۲. برو به مرحله ۱۰.

۱۳. پایان الگوریتم.

۵.آ نتیجه‌های تجربی

ما الگوریتم سی‌پی‌یو-جی‌پی‌یو خود را با استفاده از کودا-سی پیاده‌سازی نمودیم. ما این برنامه را روی سخت‌افزاری با Corei7-920 2.6 GHz و ۱۲ گیگابایت حافظه رم همراه با کارت گرافیک NVIDIA GeForce GTX 480 مبتنی بر معماری تسلا[۱۰] اجرا نمودیم. کارت گرافیک فوق دارای ۴۸۰ هسته محاسباتی با ۱.۶ گیگابایت حافظه‌ی سراسری بوده و پردازشگر آن دارای سرعت ۱.۴ گیگاهرتز است. تعداد اس‌ام‌ها در این کارت گرفیک ۱۶ عدد است. برای مقایسه، ما الگوریتم هلر که مبتنی بر سی‌پی‌یواست، را پیاده‌سازی و روی همین کامپیوتر اجرا نمودیم. برای پیاده‌سازی این الگوریتم ما از C++ و کتابخانه CGAL [۵۳] استفاده نمودیم. الگوریتم هلر را می‌توان به صورت ساده در زمان با مرتبه‌ی $O(n^3)$ پیاده‌سازی نمود. در این پیاده‌سازی حداکثر n بار باید مثلث‌بندی دلونی را انجام دهیم و در هر بار مثلث‌بندی باید n نقطه را بین مثلث‌ها توزیع کنیم. بنابراین حداکثر $O(n^3)$ زمان نیاز خواهیم داشت. با استفاده از درخت دلونی و ساخت مثلث‌بندی دلونی به صورت افزایشی [۵۴] می‌توان زمان آن را کاهش داد. ما با استفاده از کتابخانه درخت سلسله مراتبی با نام class *Triangulation_hierarchy_2* [۵۳]، این روش را در زمان $O(n^2 \log n)$ پیاده‌سازی کردیم.

برای ارزیابی، ما روش ارایه شده را بر روی داده‌ها با اندازه‌های مختلف (شامل ۶۰۰۰، ۲۵۰۰۰ و ۵۰۰۰۰ نقطه) و با اندازه خطاهای مختلف (۱۵، ۲۰، ۴۰، ۵۰، ۱۰۰ و ۲۰۰) اجرا نمودیم. همچنین برای آن که نشان دهیم که این روش بر روی داده با هر اندازه‌ای (حتی اندازه بزرگتر از حافظه جی‌پی‌یو) کار می‌کند ما آن را بر روی یک داده بزرگ شامل ۵۰۰۰۰۰ نقطه، اجرا نمودیم.

میزان حافظه موردنیاز برای نگهداری داده ساختار پیشنهادی (شامل مثلث‌ها، نقطه‌های و ...)، برای ساده‌سازی DEM با ۵۰۰۰۰۰ نقطه و بیشتر، بیش از ۲ گیگا بایت است. این میزان بیش از حافظه در دسترس بر روی سخت‌افزاری است که ما به کار گرفته‌ایم (حافظه سایر سخت‌افزارها هم در همین حدود است). بنابراین، ما داده را به چندین بخش تقسیم نموده و هر قسمت را به طور جداگانه به جی‌پی‌یو ارسال می‌کنیم. هر یک از این بخش‌ها به طور جداگانه ساده شده و سپس بخش دیگری جایگزین آنها در حافظه سراسری جی‌پی‌یو می‌شود. بعد از آن که همه بخش‌ها ساده شدند، آنها را با استفاده از الگوریتم ۶-آ با هم ادغام نموده و مجدد خطاها را محاسبه و مثلث‌بندی می‌کنیم. نتیجه‌های ما نشان می‌دهد که الگوریتم ما به طور متوسط دارای ۱۳x افزایش سرعت (نسبت به الگوریتم هلر که بر روی سی‌پی‌یو اجرا شده) بوده و برای حالتی که اندازه داده بزرگ است دارای افزایش ۴x تسریع می‌باشد. شکل‌های ۴.آ.آ، ۴.آ.ب و ۴.آ.ج مقایسه زمان اجرا برای سی‌پی‌یو، جی‌پی‌یو با ۱-اس‌ام و جی‌پی‌یو با ۱۶-اس‌ام را نشان می‌دهد.

ما هردوی این پیاده‌سازی‌ها را بر روی داده‌های DEM مربوط به ناحیه‌های Utah و Colorado [۵۵]

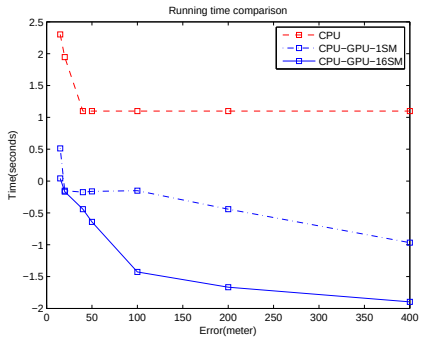
[۱۰] Tesla

اجرا نمودیم. این اجراها برای اندازه‌های مختلف خطا تکرار و نتیجه‌ها مقایسه شدند. الگوریتم ما به طور متوسط ۱۳x سریعتر از اجرای الگوریتم هلر بر روی سی‌پی‌یو است. توجه داشته باشید که با توجه به آن که برای ورودی‌های درنظر گرفته شده، حافظه‌ی جی‌پی‌یو کافی می‌باشد، ما آن را به ۱۶ قسمت مساوی تقسیم نمودیم و در یک انتقال آن‌ها را به جی‌پی‌یو فرستادیم. همان گونه که در شکل آ.۴ نشان داده شده است روش مبتنی بر سی‌پی‌یو-جی‌پی‌یو با ۱۶ اس‌ام نسبت به روش قبلی مبتنی بر سی‌پی‌یو برتری داشته و به طور متوسط ۱۳x از آن بهتر است. همچنین در شکل آ.۴.د حالتی که میزان داده بیش از حافظه‌ی جی‌پی‌یو است نشان داده شده است. برای این حالت تسریع ۴x به دست آمده است. ما برخی از خروجی‌های الگوریتم ساده‌سازی خود را به صورت سه بعدی مدل کردیم که در شکل آ.۵ آورده شده است.

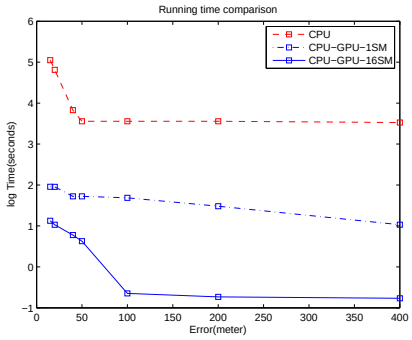
آ.۶ بحث

همان گونه که در بخش نتیجه‌های تجربی اشاره شد، ما سه پیاده‌سازی برای الگوریتم هلر انجام دادیم. ۱) پیاده‌سازی مبتنی برسی‌پی‌یو ۲)پیاده‌سازی مبتنی بر جی‌پی‌یو با ۱ اس‌ام ۳) پیاده‌سازی مبتنی بر جی‌پی‌یو با ۱۶ اس‌ام. مقایسه با روش مبتنی بر سی‌پی‌یو نشان می‌دهد آیا روش ارایه شده‌ی مبتنی بر جی‌پی‌یو بهبودی نسبت به روش قبلی دارد یا خیر. همچنین مقایسه دو روش مبتنی بر جی‌پی‌یو با ۱ اس‌ام و با ۱۶ اس‌ام به این پرسش پاسخ می‌دهد که آیا تفاوتی بین استفاده از ۱ اس‌ام با ۱۶ اس‌ام وجود دارد یا خیر و آیا زمان ادغام (که در سی‌پی‌یو انجام می‌شود) در روش مبتنی بر ۱۶ اس‌ام، باعث افزایش زمان اجرا نسبت به روش مبتنی بر ۱ اس‌ام، می‌شود یا خیر. همان گونه که از شکل آ.۴.آ، آ.۴.ب و آ.۴.ج بر می‌آید، الگوریتم مبتنی بر جی‌پی‌یو با ۱۶ اس‌ام از هر دو روش دیگر بهتر عمل می‌کند و حداقل دارای ۱۳x تسریع نسبت به الگوریتم مبتنی بر سی‌پی‌یو است. این تسریع در جاهایی که مقدار خطای ورودی کمتر است، کمتر و درجایی که مقدار خطای ورودی بیشتر است، بیشتر می‌باشد. علت این است که وقتی مقدار خطای ورودی تعیین شده کمتر است، تعداد مرحله‌های الگوریتم و مثلث‌هایی که باید ساخته شود بیشتر است. زیرا در مرحله‌های اولیه الگوریتم تعداد مثلث‌ها کم و تعداد نقطه‌های داخل هر مثلث زیاد است و به مرور تعداد مثلث‌ها بیشتر و تعداد نقطه‌های داخل هر مثلث کمتر می‌شود. از آن جایی که بیشترین موازی‌سازی بر روی نقطه‌ها انجام شده و بخش تخریب مثلث‌ها و ساخت مثلث‌های جدید در الگوریتم ۵-آ به صورت ترتیبی انجام می‌شود، کاهش خطا، باعث کاهش موازی‌سازی شده است. با این حال این الگوریتم در تمامی مرحله‌ها دارای زمان اجرای بهتری است. شکل آ.۶ میزان اشغال بودن جی‌پی‌یو را برای مجموعه داده‌های مختلف و همچنین خطاهای مختلف نشان می‌دهد. این میزان در واقع تعداد وارپ[۱۱]های فعال به بیشترین تعداد وارپ‌ها است. همان‌طور که از شکل می‌توان فهمید، میزان اشغال

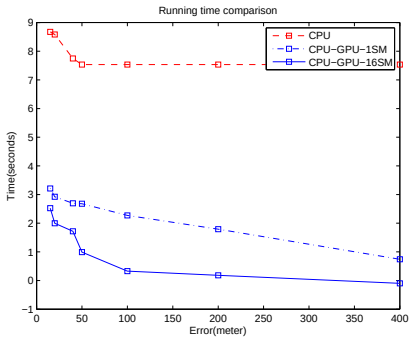
[۱۱] Warp



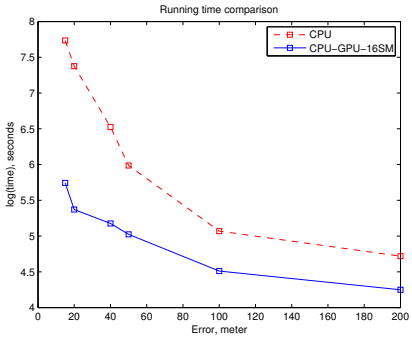
(آ) مقایسه‌ی زمان اجرای سی‌پی‌یو، جی‌پی‌یوبا یک اس‌ام و جی‌پی‌یوبا ۱۶ اس‌ام برای داده‌ی یوتا با ۶۰۰۰ نقطه.



(ب) مقایسه‌ی زمان اجرای سی‌پی‌یو، جی‌پی‌یوبا یک اس‌ام و جی‌پی‌یوبا ۱۶ اس‌ام برای داده‌ی یوتا با ۲۵۰۰۰ نقطه.

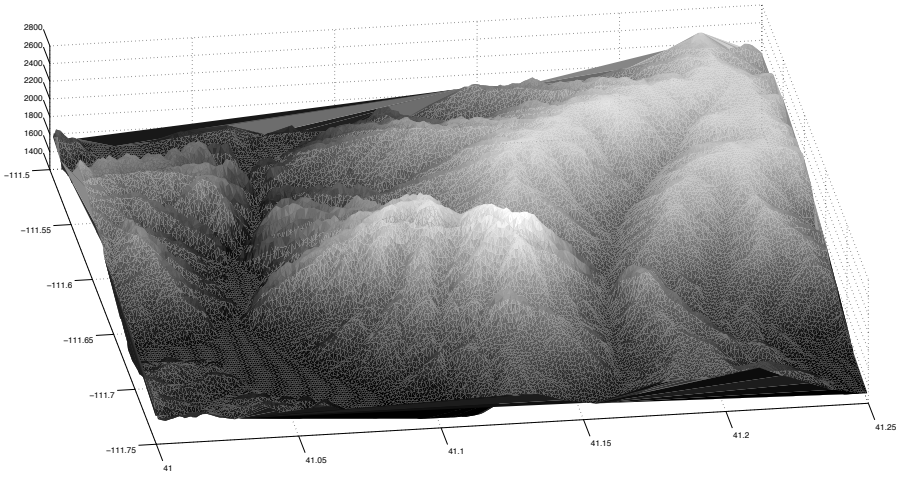


(ج) مقایسه‌ی زمان اجرای سی‌پی‌یو، جی‌پی‌یوبا یک اس‌ام و جی‌پی‌یوبا ۱۶ اس‌ام برای داده‌ی اکلاهما با ۵۰۰۰۰ نقطه.

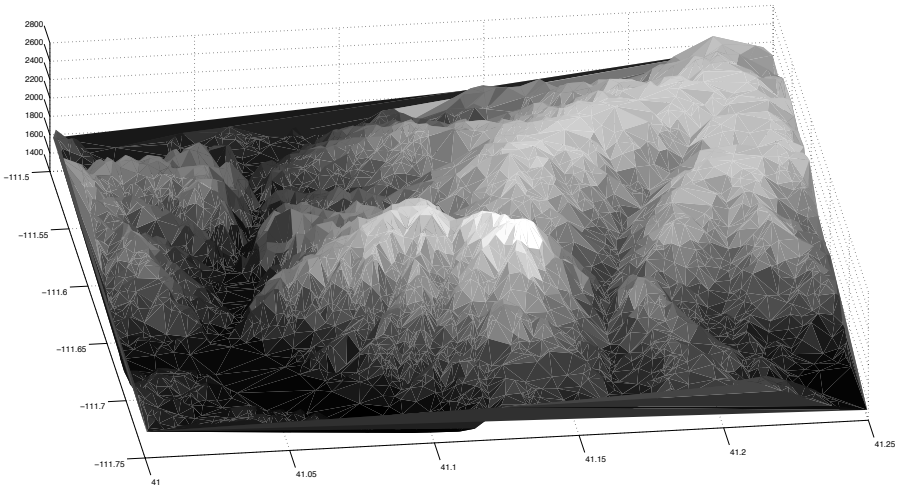


(د) مقایسه زمان اجرای سی‌پی‌یو و سی‌پی‌یو-GPU با
۱۶ اس‌ام برای داده یوتاه با ۵۰۰۰۰۰ نقطه.

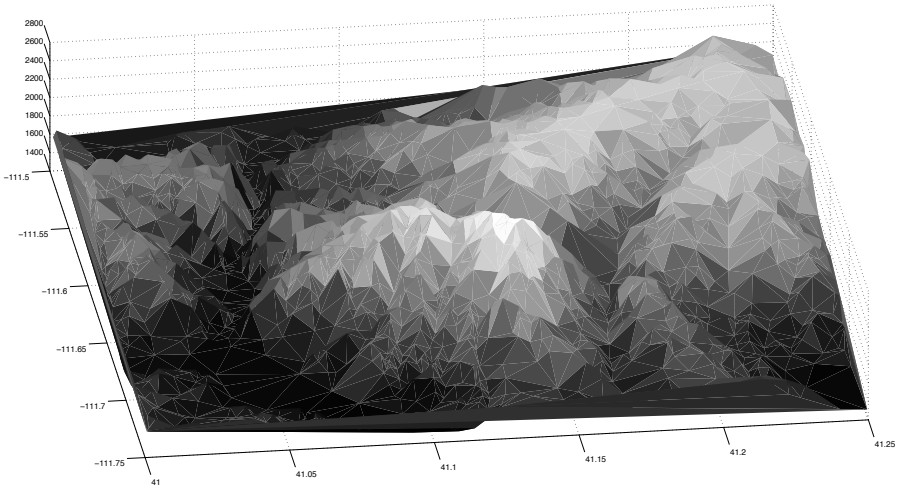
شکل آ.۴: مقایسه‌ی زمان اجرای سی‌پی‌یو و جی‌پی‌یو برای خطاهای مختلف روی داده‌های متفاوت. شکل‌های (آ)، (ب) و (ج) نتیجه‌ها را برای حالتی که داده در حافظه‌ی جی‌پی‌یو جای می‌گیرد نشان می‌دهد و شکل (د) حالتی را که چندین بار انتقال داده لازم است را نشان می‌دهد.



(آ) بخشی از زمین یوتاه با ۲۵۰۰۰ نقطه.

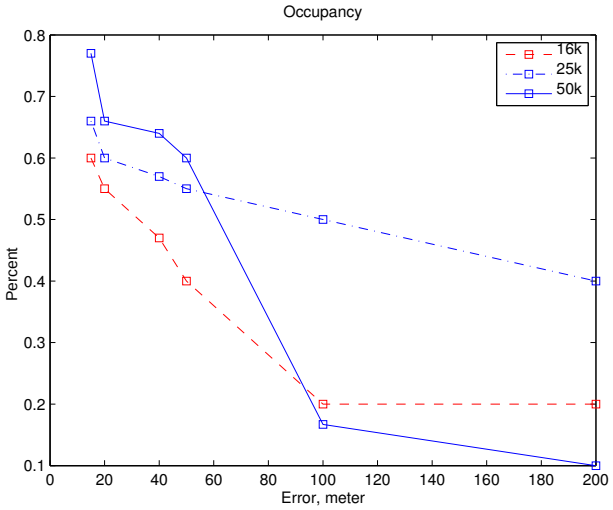


(ب) تین ساده‌شده با خطای ۲۰ متر، حاوی ۴۵۰۴ نقطه.



(ج) تین ساده شده با خطای ۵۰ متر حاوی ۲۱۱۱ نقطه.

شکل آ.۵: نمایش سه بعدی ساده‌سازی داده یوتاه با ۲۵۰۰۰ نقطه با اندازه خطاهای مختلف.



شکل آ.۶: میزان مشغول بودن جی‌پی‌یو برای خطاهای مختلف و برای داده‌های مختلف.

بودن وقتی میزان خطای کمتر است، بیشتر، و وقتی میزان خطا بیشتر است، کمتر می‌باشد.

آ.۷ جمع‌بندی

در این بخش، ما یک الگوریتم سی‌پی‌یو-جی‌پی‌یو برای ساده‌سازی زمین ارایه دادیم. روش موازی ارایه شده، اگر چه یک روش حریصانه است ولی خطای ساده‌سازی را کنترل می‌کند. روش ارایه شده، برای داده‌هایی که در حافظه‌ی جی‌پی‌یو جا می‌شوند، به طور متوسط ۱۳x تسریع، در مقایسه با الگوریتم مشابه بر روی سی‌پی‌یو، دارد. همچنین برای داده‌هایی که در حافظه جی‌پی‌یو جا نمی‌شوند این روش دارای ۴x تسریع است. به عنوان کارهای آتی می‌توان روی بهبود داده‌ساختار مورد استفاده الگوریتم در جی‌پی‌یو کار نمود. همچنین میتوان این الگوریتم را برای معماری‌های جدید چند هسته‌ای مانند زئون-فی[۱۲] بهینه نمود و زمان اجرای آن را مقایسه نمود.

[۱۲] Xeon Phi

پیوست ب

واژه‌نامه فارسی به انگلیسی

ا

ابتدایی ..... Naïve

اتصال ..... Link

اصلاح کردن ..... Rectify

ب

برازش ..... Fitting

برنامه‌ریزی پویا ..... Dynamic Programming

بی‌درنگ ..... Real-time

پ

پاره خط ..... Segment

پالایش ..... Refinement

پیمایش میان-ترتیب ..... In-order traversal

قابلیت دید ..... Visibility

پرت ..... Outlier

پرسمان ..... Query

پوسته کوژ ..... Convex Hull

پوسته کوژ وابسته ..... Relative Convex Hull

پویش‌گر ..... Scanner

ت

ترتیب جزئی ..... Partial Order

ترتیب کلی ..... Total Order

توری ..... Grid

تعاملی ..... Interactive

توپولوژی ..... Topology

توری ..... Mesh

تین ..... TIN

ج، چ، ح

جابجایی- پاره‌خط ..... Segment-dragging

جنبشی ..... Kinetic

جویباری ..... Streaming

چندضلعی ..... Ploygon

حافظه‌ی سراسری ..... Global Memory

حذف ..... Decimation

د، ز

دلونی ..... Delaunay

دنبال کردن ..... Tracking

دوبخشی ..... Bipartite

زمین ..... Terrain

زیر- مسیر ..... Subpath

س، ش، ط، غ، ف

سرشکن‌شده ..... Amortized

شاخص ..... Index

شبکه‌ی نامنظم مثلثی ..... Triangulated irregular network

شبه کد ..... Pseudo Code

طول باقی‌مانده ..... Retained Length

غیر-برخط ..... Off-line

فرشه ..... Fréchet

ک، ل

کوژ ..... Convex

لبه ..... Edge

م

محدود ..... Restricted

مکاشفه‌ای ..... Heuristic

متریک ..... Metric

مجموع زیرمجموعه ..... Subset sum

منحنی ..... Curve

میانبر ..... Shortcut

ن، و، ه، ی

ناحیه‌ی بخش‌بندی شده ..... Subdivision

نامحدود ..... Unrestricted

برازش ..... Regression

هاسدورف ..... Hausdorff

هم‌فضا ..... Homotopic

یکنوا ..... Monotone

پیوست پ

واژه‌نامه انگلیسی به فارسی

A, B

Amortized ..... سرشکن‌شده

Bipartite ..... دوبخشی

C, D

Convex ..... کوژ

Convex Hull ..... پوسته کوژ

Curve ..... منحنی

Decimation ..... حذف

Delaunay ..... دلونی

Dynamic Programming ..... برنامه‌ریزی پویا

E, F

Edge ..... لبه

Fitting ..... برازش

Fréchet ..... فرشه

G, H, I

Grid ..... توری

Global Memory ..... حافظه‌ی سراسری

هاسدورف ..... Hausdorff

مکاشفه‌ای ..... Heuristic

هم‌فضا ..... Homotopic

پیمایش میان-ترتیب ..... In-order traversal

شاخص ..... Index

تعاملی ..... Interactive

K, L

جنبشی ..... Kinetic

اتصال ..... Link

M, N

توری ..... Mesh

متریک ..... Metric

یکنوا ..... Monotone

ابتدایی ..... Naïve

O, P, Q

غیر-برخط ..... Off-line

پرت ..... Outlier

ترتیب جزئی ..... Partial Order

چندضلعی ..... Ploygon

شبه کد ..... Pseudo Code

پرسمان ..... Query

R

بی‌درنگ ..... Real-time

اصلاح ..... Rectify

پالایش ..... Refinement

برازش ..... Regression

پوسته کوژ وابسته ..... Relative Convex Hull

محدود ..... Restricted

طول باقی‌مانده ..... Retained Length

S

پویش‌گر ..... Scanner

پاره خط ..... Segment

جابجایی- پاره‌خط ..... Segment-dragging

میانبر ..... Shortcut

جویباری ..... Streaming

ناحیه‌ی بخش‌بندی شده ..... Subdivision

زیر- مسیر ..... Subpath

مجموع زیرمجموعه ..... Subset sum

T

زمین ..... Terrain

تین ..... TIN

شبکه‌ی نامنظم مثلث‌بندی شده ..... Triangulated irregular network

توپولوژی ..... Topology

ترتیب کلی ..... Total Order

دنبال کردن ..... Tracking

U, V

نامحدود ..... Unrestricted

قابلیت دید ..... Visibility

پیوست ت

فهرست مقاله‌های نویسنده

فهرست مقاله‌های پذیرفته شده در ژورنال:

1. M.A. Abam, S.Daneshpajouh, L. Deleuran, M. Ghodsi, S. Ehsani, Computing Homotopic Line Simplification, Accepted in Elsevier journal of Computational Geometry: Theory & Applications, Jan. 2014

2. S.Daneshpajouh, A.Zarei, M. Ghodsi, Path Simplification under Area Measures , Elsevier Graphical Models Journal, Volume 74, Issue 5, September 2012, Pages 283–289.

فهرست مقاله‌های در دست ارسال/ارسال شده به ژورنال:

3. S. Daneshpajouh, M. Nouri Byghi, M. Ghodsi, Computing Weak Visibility Line Simplification Inside a Simple Polygon.

4. S. Daneshpajouh, M. Ghodsi, M. Gholami, S. Keshtkar Jafari, M. Mahmoudi Aznaveh, TIN Simplification Algorithm on Graphical Processing Units, Submitted to Computers & Geosciences, 2013.

فهرست مقاله‌های پذیرفته شده در کنفرانس:

5. S. Daneshpajouh, M. Ghodsi, A Heuristic Homotopic Path Simplification Algorithm. ICCSA, Lecture Notes in Computer Science, 2011: 132–140

6. S. Daneshpajouh, M.A. Abam, L. Deleuran, M. Ghodsi, Computing Strongly Homotopic Line Simplification in the Plane. European Workshop on Computational Geometry (EuroCG), 185–188, 2011

7. S. Daneshpajouh, A.Zarei, M. Ghodsi, Path Simplification under Difference Area Measure, 14th Int'l CSI Computer Conference (CSICC'2009), Amirkabri University of Technology, Pages 276–279, July 1-2, 2009, Tehran, Iran, 2009.

8. S. Daneshpajouh, A.Zarei, M. Ghodsi, On Realistic Line Simplification under Area Measure, In proceeding of 2009 IAENG International Conference on Computer Science (ICCS-2009), Hong Kong, Pages 465–470, 18–20 March, 2009.

فهرست سایر مقاله‌ها (غیر مرتبط با پایان نامه):

9. A. Ahmadzadeh, H. Madani, K. Jafari, F. Salimi Jazi, S. Daneshpajouh, S. Gorgin, Fast and adaptive BP-based multi-core implementation for stereo matching. 10th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), pages 135–138, 2013, All around winner of Memocode design contest (Performance and Performance/Cost).

10. A. Arbabi, M. Gholami, M. Varmazyar, S. Daneshpajouh, Fast CPU-based DNA exact sequence aligner, 10th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), pages 95–98, 2012, Winner of Performance/Cost class of Memocode design contest.

11. F. Dabaghi, M. Ghodsi, S Daneshpajouh Weighted Terrain Simplification Algorithm, CSE Conference, AISP 2012.

12. S. Daneshpajouh, Mojtaba Mohammadi Nasiri, M. Ghodsi, A Fast Community Based Algorithm for Generating Crawler Seeds Set, In Proceeding of *4th International Conference on Web Information Systems and Technologies (WEBIST-2008)*, Funchal, Portugal, May 2008.

13. Hamed Yaghoobi Shahir, S. Daneshpajouh, Raman Ramsin, Improvement Strategies for Agile Software Processes , *In Proceeding of SERA 2008*, Prague, Czech Republic, August, 2008.

14. S. Daneshpajouh, M. Ghodsi, Inferring Web Core Communities, (In persian) in *2nd Conference on Information and Communication Technology Management (IRCTM)*, 2006.

15. S. Daneshpajouh, M. Ghodsi, Parallel Algorithm for Map Labeling Problem , (In persian) in *2nd Conference on Information and Knowledge Technology (IKT 2005)*, Amir Kabir University of Technology, June 2005.

مراجع

[1] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

[2] Robert B. McMaster. The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346, 1987.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[4] Howard Veregin. Line simplification, geometric distortion, and positional error. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 36(1):25–39, 1999.

[5] Wang Mao Yao Guoqing, Chen Zhun. The new triangulation-simplify algorithm of tin. *CGVR: International Conference on Computer Graphics and Virtual Reality*, 2006.

[6] Wenji Zhao, Tao Tang, Huili Gong, Fushou Duan, and Ying Mo. Dynamic data retrieval and distance decay of triangulated irregular network(tin) in three dimensional visualizations. *Geographic Information Sciences*, 12:21–26, 2006.

[7] Sumanta Guha Chansophea Chuon. Volume cost based mesh simplification. *Computer Graphics, Imaging and Visualization, International Conference on*, 0:164–169, 2009.

[8] Christian Dick, Jens Schneider, and Rüdiger Westermann. Efficient geometry compression for gpu-based decoding in realtime terrain rendering. *Computer Graphics Forum*, 28(1):67–83, 2009.

[9] Christian Dick, Jens Krüger, and Rüdiger Westermann. Gpu-aware hybrid terrain rendering. In *Computer Graphics, Visualization, Computer Vision and Image Processing*, 2010.

[10] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve-formulations and algorithms. In *Computational Morphology*, pages 71–86. North-Holland, 1988.

[11] A. Melkman and Joseph O'Rourke. On polygonal chain approximation. In Godfried T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, Amsterdam, Netherlands, 1988.

[12] Pankaj K. Agarwal, Agarwal and Kasturi R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23:273–291, 2000.

[13] W.S. Chan and Francis Chin. Approximation of polygonal curves with minimum number of line segments. In Toshihide Ibaraki, Yasuyoshi Inagaki, Kazuo Iwama, Takao Nishizeki, and Masafumi Yamashita, editors, *Algorithms and Computation*, volume 650 of *Lecture Notes in Computer Science*, pages 378–387. Springer Berlin Heidelberg, 1992.

[14] W.S. Chan and Francis Chin. Approximation of polygonal curves with minimum number of line segments. In Toshihide Ibaraki, Yasuyoshi Inagaki, Kazuo Iwama, Takao Nishizeki, and Masafumi Yamashita, editors, *Algorithms and Computation*, volume 650 of *Lecture Notes in Computer Science*, pages 378–387. Springer Berlin Heidelberg, 1992.

[15] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. In Rolf Möhring and Rajeev Raman, editors, *Algorithms - ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 29–41. Springer Berlin Heidelberg, 2002.

[16] Prosenjit Bose, Sergio Cabello, Otfried Cheong, Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Area-preserving approximations of polygonal paths. *Journal of Discrete Algorithms*, 4(4):554–566, December 2006.

[17] Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

[18] Robert B. McMaster. *A Quantitative Analysis of Mathematical Measures in Linear Simplification*. PhD thesis, Department of Geography and Meteorology, University of Kansas, Lawrence, Kansas, 1983.

[19] Mahes Visvalingam and J D Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993.

[20] Marc de Berg, Mark van Kreveld, and Stefan Schirra. A new approach to subdivision simplification. In *ACSM/ASPRS Annual Convention & Exposition Technical Papers. Bethesda: ACSM/ASPRS*, pages 79–88, 1995.

[21] Marc de Berg, Mark van Kreveld, and Stefan Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Science*, 25(4):243–257, 1998.

[22] Leonidas J. Guibas, John E. Hershberger, Joseph S.B. Mitchell, and Jack Scott Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 03(04):383–415, 1993.

[23] Regina Estkowski and Joseph S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of Annual Symposium on Computational Geometry*, SCG '01, pages 40–49, New York, NY, USA, 2001. ACM.

[24] Hiroshi Imai and Masao Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of information processing*, 09(03):159–162, 1987.

[25] Sergio Cabello, Yuanxin Liu, Andrea Mantler, and Jack Snoeyink. Testing homotopy for paths in the plane. *Discrete & Computational Geometry*, 31(1):61–81, 2004.

[26] Prosenjit Bose, Anna Lubiw, and J. Ian Munro. Efficient visibility queries in simple polygons. *Computational Geometry*, 23(3):313 – 335, 2002.

[27] M.T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, 14:445–462, 1995.

[28] John Hershberger and Jack Snoeyink. An o(nlogn) implementation of the douglas-peucker algorithm for line simplification. In *Proceeding of Annual Symposium on Computational Geometry*, pages 383–384, New York, NY, USA, 1994. ACM.

[29] John Hershberger and Jack Snoeyink. Cartographic line simplication and polygon csg formulae in o(n log* n) time. In Frank Dehne, Andrew Rau-Chaplin, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 93–103. Springer Berlin Heidelberg, 1997.

[30] Michael Godau. A natural metric for curves — computing the distance for polygonal chains and approximation algorithms. In Christian Choffrut and Matthias Jantzen, editors, *STACS*, volume 480 of *Lecture Notes in Computer Science*, pages 127–136. Springer Berlin Heidelberg, 1991.

[31] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05(01n02):75–91, 1995.

[32] Mohammad Ali Abam, Mark Berg, Peter Hachenberger, and Alireza Zarei. Streaming algorithms for line simplification. *Discrete & Computational Geometry*, 43(3):497–515, 2010.

[33] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

[34] Stefan Langerman. On the complexity of halfspace area queries. *Discrete & Computational Geometry*, 30:639–648, 2003.

[35] Sergei Bespamyatnikh. Computing homotopic shortest paths in the plane. *Journal of Algorithms*, 49(2):284 – 303, 2003.

[36] Sergei Bespamyatnikh. An optimal morphing between polylines. *International Journal of Computational Geometry and Application*, 12(3):217–228, 2002.

[37] Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, Joseph S. B. Mitchell, and T.M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002.

[38] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490 – 509, 1988.

[39] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and RobertE. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[40] Mojtaba Nouri Bygi and Mohammad Ghodsi. Weak visibility queries in simple polygons. In *Canadian Conference on Computational Geometry (CCCG)*, pages 473–478, 2011.

[41] Alireza Zarei and Mohammad Ghodsi. Efficient observer-dependent simplification in polygonal domains. *Algorithmica*, 62(3-4):842–862, 2012.

[42] Felipe Contreras. Cutting polygons and a problem on illumination of stages. Master's thesis, University of Ottawa, Canada, 1998.

[43] Boaz Ben-Moshe, Olaf A. Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In *Proceeding of Annual Symposium on Computational Geometry*, pages 27–35. ACM, 2004.

[44] Larry Palazzi and Jack Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graphical Models and Image Processing*, 56(4):304 – 310, 1994.

[45] Bernard Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3(1-4):205–221, 1988.

[46] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[47] Godfried T. Toussaint. An optimal algorithm for computing the convex hull of a set of points in a polygon. In *Proceeding of Signal Processing III: Theories and Applications.*, pages 853–856. EURASIP-86, Part 2,, 1986.

[48] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28:39–55, March 2008.

[49] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994.

[50] Pankaj K. Agarwal and Pavan K. Desikan. An efficient algorithm for terrain simplification. In *Proceedings of annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 139–147, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[51] Martin Heller. Triangulation algorithms for adaptive terrain modeling. In *Proceedings of International Symposium on Spatial Data Handling*, pages 163–174, 1991.

[52] Jay Lee. A drop heuristic conversion method for extracting irregular network for digital elevation models. *GIS/LIS '89 Proc.*, 1:30–39, November 1989.

[53] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[54] Jean Daniel Boissonnat and Monique Teillaud. The hierarchical representation of objects: the delaunay tree. In *Proceeding of Annual Symposium on Computational Geometry*, pages 260–268, New York, NY, USA, 1986. ACM.

[55] Webgis - geographic information systems resource. http://www.webgis.com/lulc_data/8_ref.html.

Abstract

In this thesis, we present different algorithms for path simplification. We first study this problem under area measures. We present an approximation algorithm for sum-area measure and an optimal algorithm for diff-area measure. Both of these algorithms can be applied on general paths. Moreover, we study the homotopic path simplification. We present a general framework for path simplification which work for desired error measure. We first present a strongly homotopic simplification algorithm for x-monotone paths. Also, we study the strongly homotopic simplification for general paths. We present a geometric tree data structure and present an algorithm based on this data structure. Furthermore, we present a homotopic simplification algorithm for general paths. To our knowledge, these two algorithms are the first optimal algorithms for general paths. For the case in which the running time is an issue, we present a heuristic method. Moreover, we study the path simplification under visibility measures. We first study this problem under the size of the weak visibility polygon and present an algorithm for max-error of the path. We prove that the simplification problem under the area of the weak visibility polygon and for sum-error of the path is NP-hard. We present an approximation algorithm for this problem.

Keywords: *Computational Geometry, Path Simplification, Homotopy, Area, Visibility*

Sharif University of Technology
Computer Engineering Department

PhD Thesis

Software Engineering

Topic

Geometric Simplification Algorithms

By
Shervin Daneshpajouh

Supervisor
Mohammad Ghodsi

February 2014