دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

رساله‌ی دکتری

مهندسی کامپیوتر

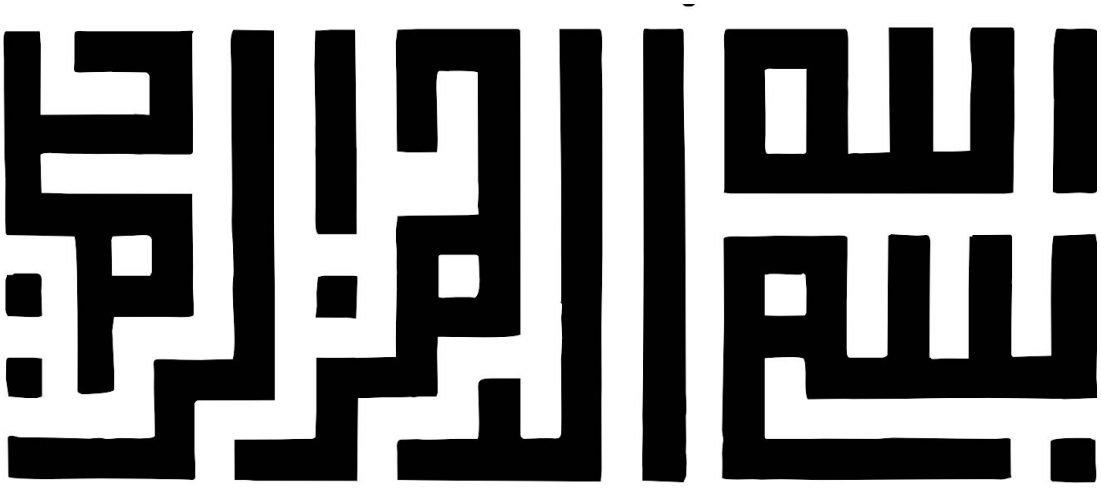
عنوان

الگوریتم‌های کارا برای آزمون دیدپذیری اشیا و شمارش آنها

نگارش
شراره علیپور

استاد راهنما
دکتر محمد قدسی

شهریور ۱۳۹۵



چکیده

محاسبه‌ی دیدپذیری در صفحه به معنای تعیین نقاطی از صفحه است که از یک ناظر واقع در صفحه دیدپذیر هستند. این مسأله کاربردهای فراوانی در گرافیک کامپیوتری، رباتیک و بازی‌های کامپیوتری دارد. در برخی از مسایل مربوط به دیدپذیری، شمارش تعداد اشیا دیدپذیر در زمان مناسب مورد نظر است که برای دسترسی به سرعت پاسخ مناسب، الگوریتم‌های فعلی به‌صورت تقریبی این شمارش را انجام می‌دهند. در این رساله مسأله‌ی آزمون دیدپذیری اشیا و شمارش تعداد اشیا دیدپذیر بررسی و مطالعه شده‌اند.

مجموعه‌ی $S = \{s_1, s_2, \ldots, s_n\}$ متشکل از n پاره‌خط دو به دو نامتقاطع و یک نقطه‌ی پرس‌وجوی p در صفحه داده شده‌اند. در مسأله‌ی دیدپذیری هدف تعیین دیدپذیری p و یک پاره‌خط $s_i \in S$ است. در مسأله‌ی شمارش نیز هدف تعیین تعداد پاره‌خط‌های دیدپذیر از p است. برای حل مسأله‌ی شمارش، دو الگوریتم تقریبی ارائه داده‌ایم. در الگوریتم اول بین حافظه و زمان پرس‌وجو و همچنین بین ضریب تقریب و زمان پرس‌وجو مصالحه ایجاد کرده‌ایم. در الگوریتم دوم روش انتخاب تصادفی به کار گرفته شده‌است. امید ریاضی زمان پرس‌وجو در بعضی حالت‌ها بهتر از الگوریتم اول است. سپس الگوریتمی تصادفی برای بدست آوردن جواب دقیق مسأله‌ی شمارش با حافظه و زمان پرس‌وجوی مشابه [۳۵] ارائه داده‌ایم. در ادامه، مسأله‌ها را در فضای ۳ بعدی در نظر گرفته‌ایم و الگوریتم‌هایی برای حل مسأله‌ها بیان کرده‌ایم. این الگوریتم‌ها را بر روی داده‌های واقعی اجرا کرده‌ایم. نتایج بدست آمده نشان داده‌اند که در عمل زمان اجرای الگوریتم‌های بیان شده و همچنین ضریب تقریب آنها کاربردی و قابل قبول است. در انتها مدل احتمالاتی از ۲ مسأله را بیان کرده‌ایم و روش‌هایی برای حل در این حالت ارائه داده‌ایم. در برخی حالت‌ها در مدل احتمالاتی ثابت کرده‌ایم که مسأله‌ی دیدپذیری احتمالاتی $\#P - complete$ است. همچنین الگوریتم‌های تقریبی برای مسأله‌ی شمارش احتمالاتی نیز ارائه داده‌ایم.

کلمات کلیدی: ۱ـ هندسه محاسباتی، ۲ـ دیدپذیر، ۳ـ الگوریتم‌های تقریبی، ۴ـ الگوریتم‌های تصادفی.

فهرست مطالب

فهرست جدول‌ها

فهرست شکل‌ها

فصل ۱

مقدمه

محاسبه‌ی دیدپذیری[۱] به‌معنای تعیین نقاطی است که توسط یک ناظر دیدپذیر باشند. این مسأله از مباحث مهم تحقیقاتی هندسه‌ی محاسباتی[۲] در سال‌های گذشته بوده‌است.

دیدپذیری در زمینه‌ی گرافیک کامپیوتری، روباتیک، بازی‌های کامپیوتری و مباحث مرتبط دیگر کاربرد دارد.

در زمینه‌ی گرافیک کامپیوتری، مسأله‌ی حذف مناطق پنهان[۳] و محاسبه‌ی سطوح دیدپذیر به‌ازای یک صحنه‌ی مجازی و یک ناظر و تعیین مناطقی که به‌طور کامل یا جزیی توسط ناظر دیدپذیر هستند، بررسی شده‌اند[۳۰].

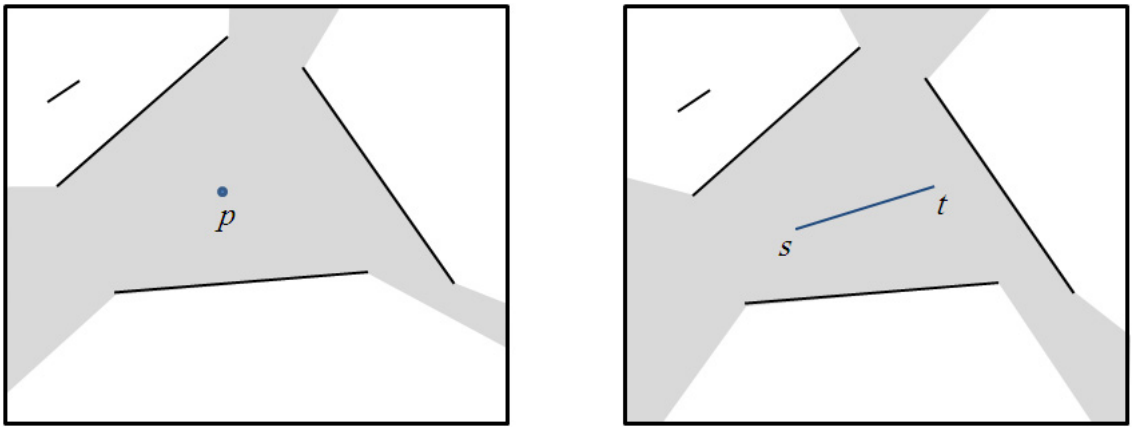
ناحیه‌ی دیدپذیر توسط یک نقطه و یا یک پاره‌خط در زمینه‌ی روباتیک، همانند عبور از موانع، برنامه‌ریزی مسیر[۴] و مکان‌یابی[۵] به‌کار می‌رود. قدرت و توانایی تعیین دیدپذیری در این است که اطلاعات مختلفی را می‌توان مستقیما از نقشه داده‌شده به‌دست‌آورد[۴۳].

تعیین دیدپذیری در صفحه، در زمینه‌ی بازی‌های کامپیوتری نیز کاربرد دارد و بدون پیشرفت در این زمینه، بازی‌های کامپیوتری مدرن گیرایی لازم را ندارند[۴۹].

به‌دلیل اهمیت موضوع، توجه ویژه‌ای به این مسأله شده‌است و راه حل‌های ترکیبیاتی و الگوریتم‌های هندسی کارا برای حل اینگونه مسأله‌ها ارائه شده‌اند.

در برخی از مسایل مربوط به دیدپذیری، شمارش تعداد اشیا دیدپذیر در زمان مناسب موردنظر است که برای دسترسی سریع به پاسخ مناسب، الگوریتم‌های فعلی به‌صورت تقریبی این شمارش را انجام

[۱] Visibility
[۲] Computational Geometry
[۳] Hidden Surface Removal
[۴] Path Planning
[۵] Localization



شکل ۱.۱: ناحیه‌ی دیدپذیر توسط یک نقطه در شکل سمت چپ و ناحیه‌ی دیدپذیر توسط یک پاره‌خط در شکل سمت راست نشان داده‌شده‌است.

می‌دهند. در این تحقیق مسأله‌ی شمارش و دیدپذیری اشیا را بررسی کرده‌ایم.

۱.۱ تعریف مسأله و نمادگذاری‌ها

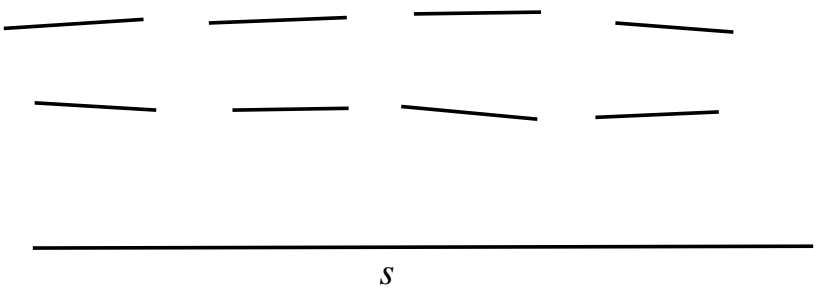
در این بخش تعریف‌ها، نمادهای مورد استفاده و همچنین مسأله‌هایی که بررسی خواهندشد، توضیح داده می‌شوند.

مجموعه‌ی $S = \{s_1, s_2, .., ..s_n\}$ شامل n پاره‌خط دوبه‌دو نامتقاطع را در فضای ۲ بعدی درنظر بگیرید. دو نقطه در صفحه را نسبت به S دیدپذیر می‌گوییم اگر پاره‌خط واصل بین آن دو هیچکدام از اعضای S را قطع نکند. همچنین نقطه‌ی p و پاره‌خط s را دیدپذیر می‌گوییم اگر نقطه‌ای مثل t عضو s وجود داشته‌باشد که t و p دیدپذیر باشند.

همچنین فرض می‌کنیم که مربع B وجود دارد که شامل تمامی پاره‌خط‌ها است. بنابراین تمامی نقاط مورد بررسی تنها داخل B در نظر گرفته می‌شوند.

با توجه به این تعریف‌ها، ناحیه‌ی دیدپذیر توسط نقطه‌ی p را مجموعه‌ی تمام نقاطی تعریف می‌کنیم که توسط p دیدپذیر هستند و ناحیه‌ی دیدپذیر توسط پاره‌خط s را مجموعه‌ی تمام نقاطی تعریف می‌کنیم که توسط حداقل یکی از نقاط پاره‌خط s دیدپذیر باشند.

ناحیه‌ی دیدپذیر توسط نقطه‌ی p را با $V_S(p)$ و ناحیه‌ی دیدپذیر توسط پاره‌خط s را با $V_S(s)$ نشان می‌دهیم. در شکل ۱.۱ ناحیه‌ی دیدپذیر توسط یک نقطه و همچنین ناحیه‌ی دیدپذیر توسط یک پاره‌خط نشان داده شده‌است. واضح است که این تعاریف را می‌توان به اشیا دیگری جز نقطه و پاره‌خط نیز تعمیم داد.



شکل ۲.۱: حالتی که در آن پیچیدگی ناحیه‌ی دیدپذیر پاره‌خط s از $O(n^4)$ است.

همان‌طور که در شکل ۱.۱ دیده‌می‌شود، ناحیه‌ی دیدپذیر توسط نقطه‌ی p به‌صورت چندضلعی ستاره‌ای شکل است که نقطه‌ی p در مرکز آن قراردارد و تعداد یال‌ها و راس‌های آن از $O(n)$ است.

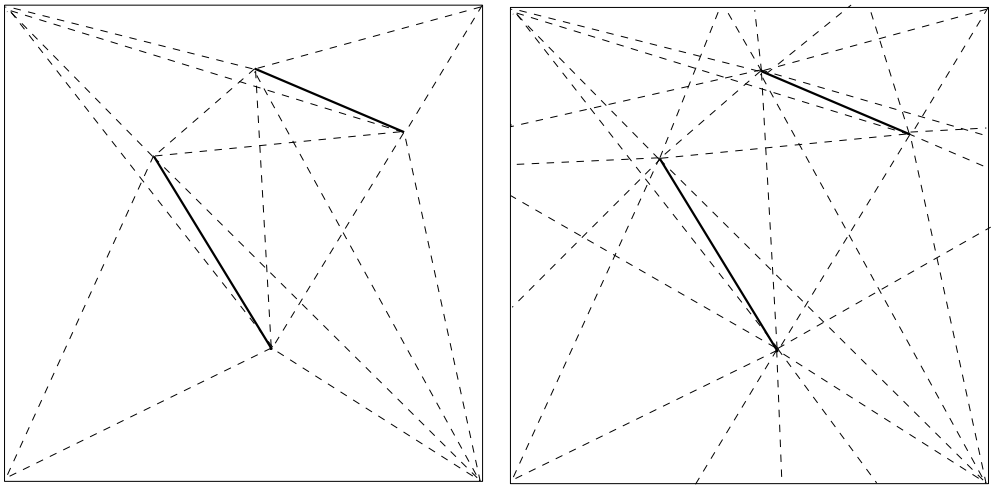
برای به‌دست آوردن ناحیه‌ی دیدپذیر توسط یک نقطه‌ی p، نقاط انتهایی پاره‌خط‌ها را به‌صورت شعاعی[۶] حول نقطه‌ی p مرتب می‌کنیم. سپس با استفاده از یک درخت جستجوی دودویی که در آن پاره‌خط‌ها به ترتیب برخورد آن‌ها با شعایی که از p شروع می‌شود، ذخیره شده‌اند، ناحیه‌ی دیدپذیر که به‌صورت مجموعه‌ای از $O(n)$ مثلث است به‌دست می‌آید. در [۵۱، ۱۴] الگوریتم محاسبه‌ی چندضلعی دیدپذیر یک نقطه توضیح داده شده‌است. چون $V_S(p)$ به‌شکل ستاره‌ای است، برای تشخیص این‌که یک نقطه پرس‌وجوی q در $V_S(p)$ قرار دارد یا نه، کافی است جستجوی دودویی انجام دهیم. در این حالت $V_S(p)$ به‌صورت داده‌ساختاری با اندازه‌ی $O(n)$ درنظر گرفته می‌شود. این داده‌ساختار می‌تواند در زمان $O(\log n)$ به این سوال پاسخ دهد که آیا q در $V_S(p)$ قرار دارد یا نه.

برخلاف ناحیه‌ی دیدپذیر یک نقطه که ساده است، ناحیه‌ی دیدپذیر یک پاره‌خط s پیچیده است و در بدترین حالت پیچیدگی آن از اندازه‌ی $\Omega(n^4)$ است. در شکل ۲.۱ این حالت نشان داده شده‌است. بنابراین فضا و زمان پیش‌پردازش زیادی برای تعیین $V_S(s)$ مورد نیاز است.

همچنین حالت‌هایی وجود دارد که برای بسیاری از اعضای $s \in S$ پیچیدگی $V_S(s)$ از $\Omega(n^2)$ است، بنابراین محاسبه‌ی جداگانه‌ی $V_S(s)$ و پیش‌پردازش آن برای مکان‌یابی[۷] نمی‌تواند داده‌ساختاری با حافظه‌ی مناسب برای پاسخ‌گویی به این سوال که آیا s و p دیدپذیر هستند یا نه، به‌دست بدهد. از این‌رو محاسبه‌ی چندضلعی دیدپذیر پاره‌خط در این حالت نیز به فضا و زمان ساخت زیادی نیاز خواهد داشت.

در ادامه مفهوم گراف‌دید[۸] و گراف‌دیدتوسعه‌یافته[۹] بیان می‌شوند.

[۶] Radial
[۷] Point Location
[۸] Visibility Graph
[۹] Extended Visibility Graph



شکل ۳.۱: در شکل سمت چپ، گراف‌دید و در شکل سمت راست، گراف‌دید توسعه‌یافته مربوط به دو پاره‌خط نشان داده شده‌است.

تعریف ۱.۱ *گراف دید: گرافی است که راس‌های آن نقاط انتهایی پاره‌خط‌های S هستند و بین دو راس یال وجود خواهد داشت اگر و تنها اگر دو راس دیدپذیر باشند و آن را با $VG(S)$ نمایش می‌دهیم.*

واضح است که تعداد یال‌های $VG(S)$ که با m نشان می‌دهیم، از مرتبه‌ی $O(n^2)$ است. بدون از دست دادن کلیت مسأله می‌توان فرض کرد که تعداد پاره‌خط‌ها از تعداد ثابتی بیشتر است. در [۳۲] الگوریتم بهینه با زمان $O(n \log n + m)$ برای محاسبه‌ی گراف‌دید مجموعه‌ای از n پاره‌خط که با یکدیگر برخوردی ندارند، ارائه شده‌است.

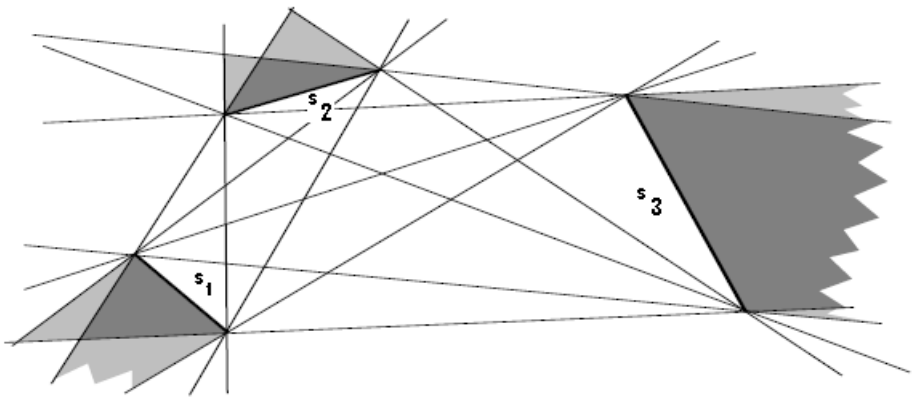
تعریف ۲.۱ *گراف دید توسعه‌یافته: از روی گراف‌دید به این‌صورت به‌دست می‌آید که به‌ازای هر یال uv در $VG(S)$ پاره‌خط e_{uv} را در دو جهت امتداد می‌دهیم تا زمانی که یکی از اعضای S و یا B (مربع شامل مجموعه‌ی S) را قطع کند. به این‌ترتیب یال‌ها و راس‌های جدیدی به گراف‌دید اضافه می‌شوند. گراف حاصل را گراف‌دید توسعه‌یافته می‌نامیم و با $EVG(S)$ نمایش می‌دهیم.*

$EVG(S)$ را نیز می‌توانیم در زمان $O(n \log n + m)$ با استفاده از الگوریتمی که در [۳۲] ارائه شده‌است، به‌دست آوریم.

در شکل ۳.۱ گراف‌دید و گراف‌دید توسعه‌یافته‌ی دو پاره‌خط نشان داده شده‌است.

تعریف ۳.۱ *افرازهای هم‌دید فضا[۱۰]: مجموعه‌ی یال‌های $EVG(S)$ و مجموعه‌ی پاره‌خط‌های S صفحه را به ناحیه‌هایی تجزیه می‌کنند که افرازهای هم‌دید فضا $VSP(S)$ نامیده می‌شود.*

[۱۰] Visibility Space Partition



شکل ۴.۱: VSP مربوط به سه پاره‌خط $S = \{s_1, s_2, s_3\}$ [۳۰].

به‌ازای هر دو نقطه‌ای که در یک ناحیه از VSP قرار دارند، مجموعه پاره‌خط‌هایی که برای آن دو پاره‌خط دیدپذیر هستند یکسان است. ناحیه‌ای از VSP که شامل نقطه‌ی p است، کلیه‌ی اطلاعات مربوط به دید در نقطه p را در بردارد. به این ترتیب که به ازای تمامی نقاط موجود در یک ناحیه از VSP ترتیب پاره‌خط‌های دیدپذیر که به صورت ساعتگرد حول هر نقطه‌ی دلخواه از آن ناحیه جاروب می‌شوند یکسان است. در شکل ۴.۱، VSP مربوط به سه پاره‌خط نشان داده شده‌است.

از آن‌جایی که $VSP(S)$ از برخورد $O(n^2)$ خط و پاره‌خط به‌دست آمده‌است، در بدترین حالت پیچیدگی آن از $O(n^4)$ است.

با انجام پیش‌پردازش بر روی پاره‌خط‌های مجموعه‌ی S می‌خواهیم به سوال‌های زیر در زمینه‌ی دیدپذیری پاسخ دهیم:

سؤال ۴.۱ *مسأله‌ی دیدپذیری*[۱۱]*: به ازای نقطه‌ی پرس‌وجوی p و پاره‌خط $s \in S$، آیا p و s همدیگر را می‌بینند؟*

سؤال ۵.۱ *مسأله‌ی شمارش*[۱۲]*: به‌ازای نقطه‌ی پرس‌وجوی p، تعداد پاره‌خط‌های دیدپذیر توسط p را به‌دست آورید.*

حل این سوال‌ها به حل سوال‌های مشابه دیگری نیز کمک خواهد کرد.

تعداد پاره‌خط‌های دیدپذیر توسط نقطه‌ی p را با m_p نشان می‌دهیم. با توجه به مطالبی که گفته شد مسأله‌ی شمارش را می‌توانیم در زمان $O(n \log n)$ و با حافظه‌ی $O(n)$ پاسخ دهیم. برای این منظور

[۱۱] Visibility Testing
[۱۲] Visibility Counting

کافی است $V_S(p)$ را به‌دست آوریم و تعداد پاره‌خط‌های مجزایی را که ناحیه‌هایی از آن‌ها به‌عنوان اضلاع $V_S(p)$ تعیین شده‌اند، محاسبه کنیم.

همچنین می‌توانیم در مرحله‌ی پیش‌پردازش به‌ازای هر ناحیه از $VSP(S)$ تعداد پاره‌خط‌های دیدپذیر در آن ناحیه را محاسبه و ذخیره کنیم. سپس در مرحله‌ی پرس‌وجو در زمان لگاریتمی ناحیه‌ای را که نقطه‌ی داده‌شده در آن قرار دارد تعیین و تعداد پاره‌خط‌های دیدپذیر از آن ناحیه را گزارش کنیم. زمان پیش‌پردازش از مرتبه‌ی $O(n^۴ \log n)$ و حافظه‌ی موردنیاز در این روش از $O(n^۴)$ خواهدبود. به‌همین ترتیب می‌توان به مسأله‌ی ۴.۱ نیز پاسخ داد.

اشکال روش اول زمان پرس‌وجوی بالا و اشکال روش دوم حافظه و زمان پیش‌پردازش زیاد است. بنابراین به دنبال یافتن روش‌هایی هستیم که از حافظه و زمان پیش‌پردازش کمتری استفاده‌کند و یا بین زمان پیش‌پردازش و زمان پاسخ‌گویی مصالحه[۱۳] ایجاد کند. استفاده از این روش‌ها منجر به یافتن پاسخ‌هایی با مقادیر تقریبی شده‌است. در روش‌های تقریبی پیچیدگی مسأله را کم می‌کنند و به این ترتیب زمان پرس‌وجو و حافظه‌ی مصرفی را بهبود می‌بخشند. ولی جوابی که به‌دست می‌آید، جواب دقیق مسأله نیست. در بسیاری از موارد نیازی به داشتن جواب دقیق مسأله نداریم و بنابراین استفاده از روش‌های تقریبی می‌تواند مفید باشد.

۲.۱ اهمیت موضوع

در این گزارش مسأله‌ی بررسی دیدپذیری بین اشیا و شمارش اشیا دیدپذیر مدنظر قرار گرفته‌اند. در ادامه خواهیم دید که روش‌های ارائه شده برای حل این مسأله‌ها منجر به حل مسأله‌های مشابه بسیاری هم در زمینه‌ی دیدپذیری و هم در زمینه‌ی شمارش اشیا، تعداد مولفه‌ها و تعداد برچسب‌ها می‌شوند. از این رو اهمیت و کاربرد حل مسأله‌هایی را که بررسی خواهند شد، بیان می‌کنیم.

یکی از کاربردهای دیدپذیری در زمینه‌ی روباتیک است. تعیین مناطق دیدپذیر پاره‌خط‌های تشکیل دهنده‌ی نقشه و همچنین استفاده از این روش‌ها به جای تکنیک ری کستینگ[۱۴] برای شبیه‌سازی خواندن حسگر [۱۵] از کاربردهای دیدپذیری در روباتیک است.

شمارش اشیا دیدپذیر در زمینه‌ی برنامه‌ریزی مسیر[۱۶] کاربرد دارد. برای مثال هنگامی که هدف تعیین اطلاعات بخصوصی در رابطه با دیدپذیری در یک نقشه است، مثلا به‌دست آوردن مناطقی که

[۱۳] Trade off
[۱۴] Ray Casting
[۱۵] Sensor Reading
[۱۶] Path planning

بیشترین تعداد اشیا را می‌بینند و یا تعیین مناطقی که یک مجموعه‌ی اشیا بخصوصی را می‌بینند. با حل این مسایل می‌توان نقشه‌های دیدپذیریی با جزییات دقیق به‌دست آورد [۳۸].

کاربرد بعدی در رابطه با تعین مسیر حرکت روبات به‌نحوی است که روبات حین حرکت با اشیا یا دیواره‌های نقشه برخورد نداشته باشد. در این حالت حرکت روبات بر مبنای اطلاعاتی که از حسگر می‌گیرد، انجام می‌پذیرد. این مسأله برخلاف برنامه‌ریزی مسیر که در آن پردازش بر روی کل نقشه انجام می‌گیرد به‌صورت محلی است و نیاز به کلیه اطلاعات محیط ندارد [۵۰].

در زمینه‌ی گرافیک کامپیوتری یکی از روش‌های نمایش اشیا ۳ بعدی برای تشخیص شی[۱۷]، محاسبه‌ی مناطق دیدپذیر شی از نقاط مختلف به‌صورت مجموعه‌ای از ناحیه‌های ۲ بعدی است که با استفاده از این مجموعه، شی مورد نظر نمایش داده می‌شود [۳۴].

به هنگام ارائه‌ی مدل‌های بزرگ و پیچیده، سخت‌افزار گرافیکی با مشکل محدودیت حافظه روبرو می‌شود. از آنجایی که در بسیاری از موارد نیازی به ارائه‌ی همه‌ی اطلاعات نداریم و تنها کافی است نواحی را در نظر بگیریم که دیده می‌شوند با استفاده از روش‌های ارائه شده در زمینه‌ی دیدپذیری، می‌توانیم مناطقی را که دیده نمی‌شوند، تشخیص دهیم و از پایپ لاین[۱۸] نمایش خارج و سرعت را بیشتر کنیم. به‌عنوان مثال در بازی‌های کامپیوتری این شیوه پیاده‌سازی شده‌است [۴۹].

دیدپذیری در زمینه‌های دیگری همچون رادیوزیتی [۱۹] ، نورهای حجمی [۲۰] و محاسبه‌ی سایه [۲۱]نیز کاربرد دارد[۴۹].

با توجه به مطالبی که گفته شد بهبود در زمان ساخت و حافظه‌ی روش‌های موجود اهمیت دارند. همچنین ارائه‌ی الگوریتم‌هایی که به‌راحتی قابل پیاده‌سازی باشند نیز مهم است.

در فصل‌های آینده‌ی، الگوریتم‌های تقریبی و تصادفی نیز برای حل این مسأله‌ها بیان شده‌اند. از آنجایی که جواب‌های تقریبی نیز کاربردی هستند، ارائه‌ی این روش‌ها در صورتی‌که بتوان زمان و حافظه‌ی الگوریتم را بهبود بخشند، دارای اهمیت خواهدبود.

مسأله‌های مربوط به دیدپذیری در فضاهای بزرگتر از ۲ بعد کاربردی‌تر هستند و ارائه‌ی روش‌های کارا در این فضاها، در زمینه‌های عنوان شده کمک بسیاری خواهد کرد. به‌عنوان مثال دیدپذیری در تین‌ها [۲۲]، در مباحث جغرافیایی کاربرد دارد.

در ادامه خواهیم دید روش‌هایی که برای حل مسایل شمارش اشیا دیدپذیر عنوان می‌شوند رابطه‌ی

[۱۷] Object Recongnition
[۱۸] Pipeline
[۱۹] Radiosity
[۲۰] Volumetric lights
[۲۱] Shadow recognition
[۲۲] Triangulated Irregular Networks(TINs)

مستقیمی با مسایلی در زمینه‌ی شمارش تعداد مولفه‌ها، تعیین تعداد برچسب‌ها و داده‌ساختارهایی همچون درخت‌افراز دارند. مفاهیم عنوان شده نیز کاربردهای فراوانی در گرافیک کامپیوتری دارند.

۳.۱ نتایج به‌دست آمده

در این تحقیق به بررسی روش‌های حل مسأله‌های عنوان‌شده پرداخته‌ایم و الگوریتم‌های جدیدی ارائه‌کرده‌ایم. همچنین روش‌های مختلفی را که به‌کار برده‌ایم، توضیح داده‌ایم و مسایل متعددی را که حل آنها منجر به حل مسأله‌ی ما و مسایل مشابه دیگر می‌شود، بیان کرده‌ایم.

برای حل مسأله‌ی شمارش، ابتدا الگوریتمی تقریبی ارائه داده‌ایم به‌طوریکه به ازای هر $0 < \delta \leq 1$ و $0 \leq \beta \leq \frac{2}{3}$ مسأله‌ی شمارش را در زمان مورد انتظار[۲۳] $O(\frac{1}{\delta^3} m^{\beta/2} \log m)$ پرس‌وجوی $O(m^{2-3\beta/2} \log m)$ و حافظه‌ی مورد انتظار $O(m^{2-3\beta/2})$ حل می‌کند. این الگوریتم مقدار m'_p را به عنوان خروجی برمی‌گرداند به‌طوری که اگر $m_p \geq \frac{1}{\delta^3} m^{\beta/2} \log m$، با احتمال حداقل $1 - \frac{1}{\log m}$ خواهیم داشت: $m_p \leq m'_p \leq (1+\delta)m_p$ و در صورتی‌که $m_p < \frac{1}{\delta^3} m^{\beta/2} \log m$، مقدار دقیق m_p برگردانده می‌شود. دقت شود که هرچه تعداد پاره‌خط‌های مجموعه‌ی S بیشتر شود، $1 - \frac{1}{\log m}$ به یک نزدیکتر می‌شود. نتایج بدست آمده در [۸] به چاپ رسیده است. سپس الگوریتم تقریبی تصادفی دیگری ارائه داده‌ایم که با امید ریاضی زمان پیش‌پردازش و حافظه‌ی $O_\epsilon(n^2)$ و $O(n^2)$ می‌تواند مسأله‌ی شمارش را در زمان مورد انتظار $O_\epsilon(\frac{1}{\delta^2}\sqrt{n})$ پاسخ دهد. این الگوریتم در صورتی‌که $m_p < \sqrt{n}$ باشد مقدار دقیق جواب مسأله را بدست می‌آورد و در غیر این‌صورت m'_p را بر می‌گرداند به‌طوری‌که با احتمال نزدیک به ۱ خواهیم داشت:

$$(1-\delta)m_p \leq m'_p \leq (1+\delta)m_p.$$

در این‌جا و در ادامه‌ی پایان‌نامه $\epsilon > 0$ و $\delta > 0$ ثابت‌هایی هستند که می‌توان به‌اندازه‌ی دلخواه کوچک انتخاب کرد و همچنین داریم:

$$O_\epsilon(f(n)) = O(f(n)n^\epsilon).$$

نتایج بدست آمده در [۹] به چاپ رسیده است. سپس الگوریتمی تصادفی برای بدست آوردن جواب دقیق مسأله با حافظه و زمان پرس‌وجوی مشابه [۳۵] ارائه داده‌ایم. در ادامه مسأله را در فضای ۳ بعدی در نظر گرفته‌ایم و الگوریتم‌هایی برای حل مسأله‌ها بیان کرده‌ایم. این الگوریتم‌ها را بر روی

[۲۳] Expected time

داده‌های واقعی اجرا کرده‌ایم و نتایج بدست آمده نشان داده‌اند که در عمل زمان اجرای الگوریتم‌های بیان شده و همچنین ضریب تقریب آنها کاربردی و قابل قبول است. نتایج این بخش در [۱۰] به چاپ رسیده است. در انتها مدل احتمالاتی از ۲ مسأله را بیان کرده‌ایم و روش‌های برای حل در این حالت ارائه داده‌ایم. در برخی حالت‌ها در مدل احتمالاتی ثابت کرده‌ایم که مسأله $\#P - complete$ است. همچنین الگوریتم‌های تقریبی نیز ارائه داده‌ایم. نتایج این قسمت نیز در [۱] به چاپ رسیده است. ایده‌های استفاده شده در [۸] و [۹] در حل مسایل مشابه نیز به کار رفته است (مراجعه شود به [۴۶]).

۴.۱ ساختار پایان‌نامه

در فصل ۲ کارها و نتیجه‌های پیشین و داده‌ساختارهایی که در روش‌های پیشین استفاده شده‌اند، توضیح داده می‌شوند. این فصل شامل مطالبی در رابطه با ادبیات موضوع و نتایج مرتبط با مسأله‌های مطرح شده در این پایان نامه است. قسمتی از این نتایج در فصلهای آینده استفاده خواهند شد. همچنین در رابطه با داده‌ساختار درخت افراز نیز مطالبی عنوان شده‌است. این داده‌ساختار کاربردهای فراوانی در رابطه با مسأله‌های مورد بررسی ما دارند. مطالعه‌ی دقیق این داده‌ساختار در ادامه به حل مسأله‌ها کمک می‌کند.

در فصل‌های ۳ و ۴ نتیجه‌های جدید به‌دست‌آمده را برای حالت ۲ بعدی مسأله بیان می‌کنیم. در فصل ۳ ابتدا مسأله را به صورت مدل ریاضی گرافی تعریف کرده ایم. به این ترتیب که به ازای هر نقطه ی پرسوجو ابتدا گراف مسطحی نسبت می دهیم، سپس ثابت می کنیم که تعداد وجه های گراف منهای ۱ یا ۲ مساوی تعداد پاره خط هایی هستند که دید نمی شوند. با استفاده از فرمول اویلر برای گراف های مسطح روشی برای محاسبه ی تعداد وجه ها ارائه داده ایم. برای این منظور تعداد وجه ها و یال های گراف را با الگوریتم تصادفی به طور تقریبی محاسبه می کنیم.

در فصل ۴ ابتدا الگوریتم تصادفی دیگری برای مسأله‌ی شمارش ارائه می‌دهیم. اساس این الگوریتم مشابه فصل پیشین است ولی تفاوت‌هایی نیز دارد. سپس در ادامه الگوریتم دقیقی برای مسأله‌ی شمارش ارائه می‌کنیم. در این الگوریتم از داده‌ساختار درخت افراز نیز با اعمال تغییراتی استفاده می‌کنیم.

در فصل ۵ مسأله را در فضای ۳ بعدی بیان می‌کنیم. برخی قضیه‌ها که در فضای ۲ بعدی بیان شده بودند را به فضای ۳ بعدی تعمیم می‌دهیم. الگوریتم‌های بیان شده در این فصل مشابه فصل‌های گذشته، تصادفی یا تقریبی هستند. همچنین برای بررسی کارایی این الگوریتم‌ها، آنها را بر روی داده‌های واقعی اجرا می‌کنیم. نتایج تجربی نشان می‌دهند که کارایی این الگوریتم مناسب است.

فصل ۶ نیز در رابطه با مدل احتمالاتی مسأله است. در این فصل با کاهش مسأله‌ی شمارش تعداد تطابق‌های کامل گراف ۲ بخشی به مسأله‌ی دید احتمالاتی ثابت می‌کنیم که این مسأله نیز $\#P-complete$ است. سپس در حالت خاصی، الگوریتمی بازگشتی برای حل مسأله‌ی دید احتمالاتی ارائه می‌دهیم. در این روش از درخت جستجوی دودویی برای بهبود زمان و حافظه‌ی الگوریتم بازگشتی استفاده می‌کنیم. در ادامه الگوریتمی بیان می‌کنیم که مسأله‌ی شمارش احتمالاتی را با ضریب تقریب ۲ حل می‌کند.

در هر کدام از این فصل‌ها روش ارائه‌شده برای حل مسأله را توضیح می‌دهیم. سپس نتیجه‌های به‌دست آمده را بیان می‌کنیم و در انتهای هر فصل ایده‌ها، مسأله‌ها و روش‌های پیشنهادی در رابطه با ادامه‌ی تحقیق را برمبنای روش جدید بیان می‌کنیم. فصل ۷ نیز شامل نتیجه‌گیری کلی و پیشنهادهایی برای ادامه‌ی تحقیق است.

فصل ۲

مروری بر ادبیات موضوع و داده‌ساختارهای موجود

در این فصل نتیجه‌های مقاله‌های مختلف در رابطه با مسایل مطرح شده را توضیح می‌دهیم. همچنین داده‌ساختارهای مورد استفاده و قضیه‌های پایه‌ای را که در فصل‌های بعدی استفاده می‌شوند، بیان می‌کنیم.

۱.۲ مروری بر روش‌های پیشین

در این قسمت ادبیات موضوع و نتایج پیشین در ۲ بعد و سپس نتایجی در رابطه با دیدپذیری در فضاهای با بعد بزرگتر از ۲ را بیان می‌کنیم.

۱.۱.۲ مروری بر روش‌های پیشین در ۲ بعد

تحقیقات فراوانی در رابطه با مسأله‌ی دیدپذیری در صفحه انجام شده‌است و مقاله‌های بسیاری در این زمینه وجود دارند[۱۲، ۱۳، ۱۸، ۱۹، ۴۷، ۵۶]. همچنین مروری اجمالی در رابطه با مسأله‌ی دیدپذیری در [۳۳] جمع‌آوری شده‌است. در این بخش تعدادی از نتیجه‌های مرتبط با مسأله‌ی مورد بحث در این تحقیق بیان می‌شود.

برای حل مسأله در حالت‌های خاص روش‌هایی ارائه شده‌است. در [۳۲] نشان داده‌اند اگر پاره‌خط‌های مجموعه‌ی S تشکیل یک چندضلعی بدهند در این‌صورت پیچیدگی $VSP(S)$ از $O(n^۳)$ خواهدبود. در این حالت مسأله را می‌توان با زمان پیش‌پردازش $O(n^۳)$ و حافظه‌ی $O(n^۳)$ در زمان

$O(\log n)$ پاسخ داد. همچنین در [۱۲] برای این حالت خاص، الگوریتمی با حافظه‌ی موردنیاز $O(n^۲)$ و زمان پرس‌وجوی $O(\log^۲ n)$ ارائه شده‌است.

در [۱۵] نیز با زمان پیش‌پردازش و حافظه‌ی $O(n^۲)$ می‌توان $V_S(p)$ را در زمان $O(n)$ گزارش کرد. در [۵۳] با زمان پیش‌پردازش $O(n^۲ \log n)$ و حافظه‌ی $O(n^۲)$ الگوریتمی ارائه شده‌است که $V_S(p)$ را در زمان $O(|V_S(p)| \log(n/|V_S(p)|))$ گزارش می‌کند. $|A|$ نشان دهنده‌ی تعداد اعضای مجموعه‌ی A است.

در [۴۷] الگوریتمی با فضای $O(m)$ ارائه شده‌است که $V_S(p)$ را در زمان $O(|V_S(p)| \log n)$ محاسبه می‌کند.

هنگامی که پاره‌خط‌های S به شکل یک چندضلعی با h حفره هستند، در [۵۶] الگوریتمی با فضای $O(n^۳)$ ارائه شده‌است که می‌تواند $V_S(p)$ را با زمان پیش‌پردازش $O(m_p \log n)$ محاسبه کند و زمان پرس‌وجو از $O(min\{h, m_p\} \log n + m_p)$ است.

در [۳۰، ۳۱] برای مسأله‌ی ۵.۱ دو الگوریتم ارائه کرده‌اند. اولین الگوریتم از داده ساختاری با اندازه‌ی $O((m/r)^۲)$ استفاده می‌کند که در زمان $O(\log n)$ جواب تقریبی مسأله را با خطای r به‌دست می‌آورد و $۱ \leq r \leq m$.همچنین با الگوریتمی دیگر با داده ساختاری با اندازه‌ی $O((m^۲ \log^{O(۱)} n)/l)$ و زمان پرس‌وجوی $O(l \log^{O(۱)} n)$ جواب تقریبی مسأله را با مقدار خطای δn به‌ازای هر $\delta > ۰$ به‌دست می‌آورند که $۱ \leq l \leq n$.

در [۳۵] برای حل مسأله‌ی شمارش الگوریتمی تقریبی با زمان پیش‌پردازش و حافظه‌ی $O_\epsilon(m^{۱+\alpha}) =$ $O_\epsilon(n^{۲(۱+\alpha)})$ و زمان پرس‌وجوی $O_\epsilon(m^{(۱/۲)(۱-\alpha)}) = O_\epsilon(n^{۱-\alpha})$ ارائه شده‌است. ضریب تقریب این الگوریتم ۲ است.

ایده‌ای که در [۳۵] به‌کارگرفته شده‌است محاسبه‌ی $V_S(s)$ به‌صورت اجتماعی از مثلث‌ها است. حل مسأله در این حالت معادل تعیین مثلث‌های شامل یک نقطه‌ی داده شده‌است. از آنجایی که مثلث‌های مربوط به $V_S(s)$ اشتراک دارند بنابراین در برخی حالت‌ها مثلث‌های شامل نقطه‌ی داده‌شده مربوط به یک پاره‌خط بیش از یک‌بار شمرده می‌شوند. اثبات شده‌است که تعداد مثلث‌های شامل یک نقطه حداکثر ۲ برابر مقدار اصلی جواب است و بنابراین الگوریتمی تقریبی با ضریب تقریب ۲ ارائه داده‌اند.

در [۴۵] الگوریتمی ارائه شده‌است که با حافظه‌ی $O(m)$ و زمان پیش‌پردازش $O(m \log(\sqrt{m}/n))$ که $n^۲ \leq m \leq n^۴$، می‌توان $V_S(p)$ را در زمان $O(n^۲ \log(\sqrt{m}/n)/\sqrt{m})$ به‌دست آورد و اگر بخواهیم $V_S(p)$ را گزارش کنیم، $|V_S(p)|$ نیز به زمان پرس‌وجو اضافه می‌شود. با استفاده از این الگوریتم هم می‌توان به مسأله‌ی شمارش با ضریب تقریب ۲ پاسخ داد.

در [۱۱] نیز اثبات شده‌است که تعداد نقاط انتهایی دیدپذیر پاره‌خط‌ها که با ve_p نشان می‌دهیم، حداکثر ۲ برابر تعداد پاره‌خط‌های دیدپذیر است. پس کافی است به جای محاسبه‌ی منطقه‌ی دیدپذیر یک پاره‌خط، منطقه‌ی دیدپذیر نقطه‌های انتهایی پاره‌خط‌ها را به‌دست آوریم. به‌این ترتیب می‌توانیم جواب مسأله را با ضریب تقریب ۲ به‌دست آوریم. با استفاده از این روش، پیچیدگی راه حل ارائه شده در [۳۵] کاسته می‌شود.

۲.۱.۲ مروری بر روش‌های پیشین در فضاهای با بعد بزرگتر از ۲

در این قسمت نتایجی در رابطه با دیدپذیری در فضاهای با بعد بزرگتر از ۲ ارائه می‌شوند.

در [۴۸] نتایجی در رابطه با دیدپذیری در حالتی که هدف بررسی دیدپذیری یک نقطه بر روی یک زمینه [۱] است، بیان شده‌است. زمینه، سطحی چندوجهی است که با تعیین ارتفاع برای راس‌های یک سطح مثلث‌بندی شده به‌وجود آمده است. حالت‌های مختلفی از مسأله در نظر گرفته شده‌است. روشی ارائه شده‌است که با زمان و حافظه‌ی $O(n\alpha(n)\log n)$ می‌توان تعین کرد که آیا ۲ نقطه‌ی p و q بر روی زمینه دیدپذیر هستند یا نه (که $\alpha(n)$ عکس تابع آکرمن [۲] است). همچنین مسأله را برای حالتی که p بر روی یک خط راست حرکت می‌کند در نظر گرفته‌اند.

در [۵۴] به‌ازای مجموعه‌ای از n مثلث دو به دو نامتقاطع در فضای ۳ بعدی، ثابت شده‌است پیچیدگی منطقه‌ی دیدپذیر یک مثلث از مثلث دیگر از مرتبه‌ی $O(n^۶)$ است و همچنین در صورتی‌که مجموعه‌ی مثلث‌ها تشکیل یک زمینه بدهند از مرتبه‌ی $O(n^۴)$ خواهدبود.

در [۴۲] نتایج به‌دست آمده در [۵۴] بهبود داده شده‌اند. آنها ثابت کرده‌اند که پیچیدگی منطقه‌ی دیدپذیر یک پاره‌خط برای حالت اول از $O(n^۵)$ و برای یک مثلث از $\Theta(n^۷)$ است و در حالت دوم، برای یک پاره‌خط و یک مثلث از $\Omega(n^۴)$ و $O(n^۵)$ است. در [۱۶] نیز روشی تقریبی برای محاسبه‌ی منطقه‌ی دیدپذیر یک نقطه بر روی یک زمینه ارائه شده‌است که به سوالاتی همانند تعیین دیدپذیری ۲ نقطه بر روی زمینه پاسخ می‌دهد. در این مقاله الگوریتم ارائه شده با روش‌های پیشین مقایسه شده است.

در [۲۸] نیز حالت خاصی از مسأله بررسی و ثابت شده‌است که اگر مثلث‌های تشکیل‌دهنده‌ی زمینه دارای خصوصیات زیر باشند:

۱. اندازه ی کوچکترین زاویه‌ی هر مثلث در یک بازه‌ی تعیین‌شده باشد.

[۱] Terrain

[۲] Inverse Ackermann Function

۲ . اندازه‌ی زاویه بین هر مثلث و خط افقی در یک بازه‌ی تعیین‌شده باشد.

۳. حاصل تقسیم طول بلندترین یال مثلث‌ها بر طول کوتاهترین یال مثلث‌ها در یک بازه‌ی تعیین‌شده باشد.

در این‌صورت پیچیدگی نقشه‌ی دید [۳] زمینه در بدترین حالت از مرتبه‌ی $O(n^۲)$ و امید ریاضی پیچیدگی زمینه از $\Theta(n)$ خواهدبود. نقشه‌ی دید زمینه نسبت به نقطه‌ی p، تصویر قسمت‌های دیدپذیر مثلث‌های زمینه بروی صفحه‌ی دید است.

۲.۲ کاربرد درخت افراز در مسأله‌ی شمارش

همان‌طور که در بخش قبلی عنوان کردیم با استفاده از روش ارائه شده در [۳۵] حل مسأله‌ی شمارش معادل خواهدبود با انجام پیش‌پردازش برروی تعدادی مثلث و به‌دست آوردن تعداد مثلث‌های شامل نقطه‌ی پرس‌وجو. برای این منظور از داده‌ساختاری به نام درخت‌افراز [۴] استفاده می‌شود. در ادامه، نحوه‌ی استفاده از این داده‌ساختار توضیح داده می‌شود.

۱.۲.۲ داده‌ساختار برای شمارش مثلث‌ها

مجموعه‌ی T از مثلث‌ها در صفحه داده شده‌است. می‌خواهیم بر روی این مثلث‌ها پیش‌پردازش انجام دهیم به‌طوری‌که تعداد مثلث‌های شامل یک نقطه‌ی داده‌شده را به‌دست آوریم.

فرض کنید Δ یک مثلث باشد، در این‌صورت Δ از برخورد چهار خط $h(\Delta) = (u_۱, u_۲, d_۱, d_۲)$ تشکیل شده‌است که این چهار خط، $u_۱$ و $u_۲$ از بالا و $d_۱$ و $d_۲$ از پایین Δ را احاطه می‌کنند و در این حالت $u_۱ = u_۲$ یا $d_۱ = d_۲$ خواهدبود.

با استفاده از نگاشت استاندارد دوگان [۲۷]، چهار خط $h(\Delta)$ به چهار نقطه‌ی ${u_۱}^*, {u_۲}^*, {d_۱}^*, {d_۲}^*$ نگاشت می‌شوند و نقطه‌ی p نیز به خط p^* نگاشت می‌شود.

نقطه‌ی p داخل Δ خواهدبود اگر و تنها اگر خط p^* بالای ${u_۱}^*$ و ${u_۲}^*$ و زیر ${d_۱}^*$ و ${d_۲}^*$ باشد. قرار می‌دهیم:

$$h^*(\Delta) = ({u_۱}^*, {u_۲}^*, {d_۱}^*, {d_۲}^*)$$

[۳] Visibility Map
[۴] Partition Tree

ساختاری که برای شمارش مثلث‌های T به‌کار می‌رود نقاط هشت بعدی به‌صورت:

$$h^*(T) = \{h^*(\Delta) : \Delta \in T\}$$

را ذخیره می‌کند. به‌ازای نقطه‌ی پرس‌وجوی p می‌خواهیم تعداد نقاط $(a, b, c, d) \in h^*(T)$ را که در شرایط زیر صدق می‌کنند، به‌دست آوریم.

۱. a بالای p^*

۲. b بالای p^*

۳. c پایین p^*

۴. d پایین p^*

شمارش تعداد نقاطی که هریک از شرایط را بررسی می‌کند معادل خواهد بود با حل مسأله‌ی جستجوی بازه‌ی نیم‌فضا[۵]. داده‌ساختارهای مختلفی برای حل این مسأله وجود دارند. در مسأله‌ی شمارش تعداد مثلث‌ها، می‌توان از داده‌ساختاری به نام درخت‌افراز چهار لایه با زمان پیش‌پردازش و فضای $O_\epsilon(k)$ و زمان پرس‌وجوی $O_\epsilon(n/\sqrt{k})$ استفاده کرد. این داده‌ساختار می‌تواند تعداد اعضای $h^*(T)$ را که در شرط‌های بالا صدق می‌کنند، برای هر p^* به‌دست آورد.

۲.۲.۲ جستجوی بازه‌ی نیم‌فضا و درخت‌افراز

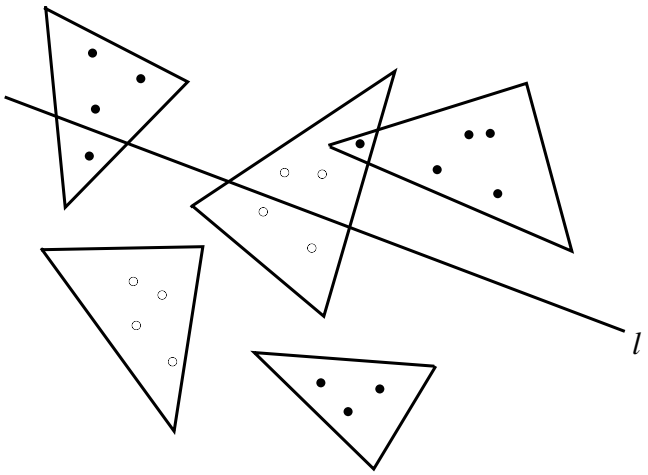
مسأله‌ی جستجوی بازه‌ی نیم‌فضا به این‌صورت بیان می‌شود.

سؤال ۱.۲ *تعداد n نقطه در فضا داده شده‌است. می‌خواهیم بر روی این نقاط پیش‌پردازش انجام دهیم به‌طوری‌که به‌ازای یک صفحه‌ی داده‌شده تعداد نقاط بالای آن صفحه را به‌دست آوریم.*

این مسأله در مقاله‌های بسیاری بررسی شده‌است[۶، ۴، ۲، ۲۵]. در این‌جا از درخت‌افراز برای پاسخ‌گویی به این مسأله استفاده می‌کنیم.

تعریف ۲.۲ *به‌ازای n نقطه‌ی داده‌شده در صفحه می‌خواهیم صفحه را به مناطق مختلف تقسیم کنیم به‌طوری‌که هرکدام از مناطق حداکثر شامل تعدادی معین از نقاط باشند و همچنین هر خطی تعداد مشخصی از این مناطق را قطع کند(شکل ۱.۲). در این‌صورت به‌ازای یک خط داده‌شده، تعدادی از مناطق به‌طور کامل بالا و یا پایین آن خط خواهندبود و بقیه توسط آن خط قطع می‌شوند. به‌ازای*

[۵] Halfspace Range Searching



شکل ۲.۱ : مثلث‌هایی که هرکدام از آنها تعداد معینی از نقاط را در بردارند و هر خط دلخواه l تعداد مشخصی از آنها را قطع می‌کند. دقت شود که مثلث‌ها ممکن است که اشتراک نیز داشته باشند.

مناطقی که به‌طور کامل بالای آن خط قرار دارند نقاط داخل آن مناطق نیز بالای آن خط خواهند بود. و به ازای مناطقی که توسط آن خط قطع می‌شوند، الگوریتم را به‌صورت بازگشتی اجرا می‌کنیم (شکل ۲.۲). داده ساختاری با خصوصیات گفته شده را ***درخت‌افراز*** *می‌نامیم.*

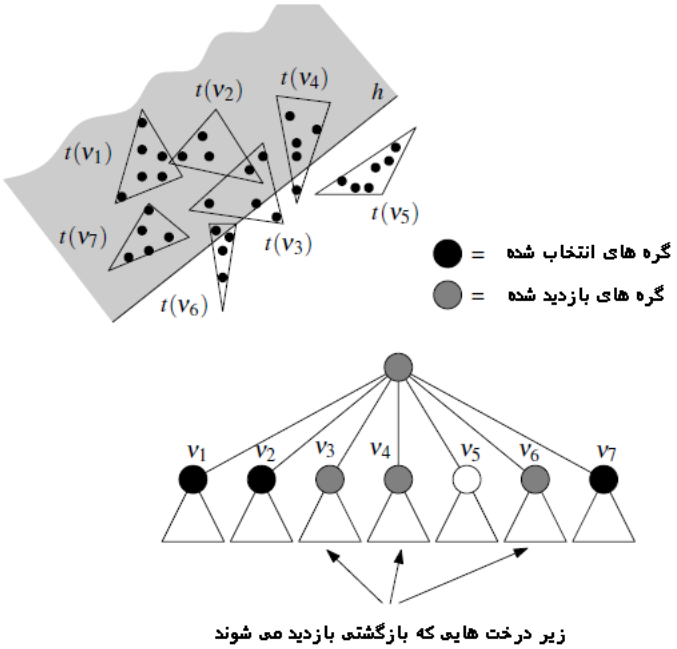
کارایی این روش به پارامترهای استفاده شده در ساخت درخت افراز بستگی دارد. برای جزییات بیشتر در رابطه با نحوه‌ی ساخت درخت‌افراز به [۲۶، ۲۳، ۴۰، ۵۵] مراجعه شود. با استفاده از درخت افراز خواهیم داشت:

قضیه ۳.۲ ([۴۰]) *به‌ازای n نقطه‌ی داده‌شده در فضای d بعدی می‌توان با پیش‌پردازش در زمان $O(n\log n)$ و حافظه‌ی $O(n)$ در زمان $O(n^{1-1/d}(\log n)^{O(1)})$ به مسأله‌ی جستجوی بازه‌ی نیم‌فضا پاسخ داد.*

همچنین با استفاده از $(1/r)-cutting$ که در ادامه بیان می‌شود، می‌توانیم بین حافظه‌ی مصرفی و زمان پرس‌وجو مصالحه ایجاد کنیم. فرض کنید L مجموعه‌ای از n خط در صفحه باشد و $r \leq n$ یک پارامتر داده شده باشد. یک $(1/r)-cutting$ برای L، مجموعه‌ای از مثلث‌هایی است که نقاط داخلشان با یکدیگر اشتراک ندارند، به طوریکه تمام صفحه پوشش داده می‌شود و هر مثلث حداکثر $\frac{n}{r}$ از خط‌های مجموعه‌ی L را قطع می‌کند.

قضیه ۴.۲ ([۴۱]) *با زمان پیش‌پردازش $O(nr)$ و حافظه‌ی $O(r^2)$ می‌توان یک $(1/r)-cutting$ به دست آورد.*

در این‌صورت خواهیم داشت:



شکل ۲.۲: یک درخت‌افراز و درخت جستجوی معادل آن. جستجو در درخت به ازای نیم‌صفحه‌ی h، نشان داده شده‌است[۲۷].

قضیه ۵.۲ ([۴۰]) *به‌ازای n نقطه‌ی داده شده در فضای d بعدی و پارامتر m، $n \leq m \leq n^d$، می‌توان با زمان پیش‌پردازش $O_\epsilon(m)$ و حافظه‌ی $O_\epsilon(m)$، در زمان $O(n(\log n)^{O(1)}/m^{1/d})$ به مسأله‌ی جستجوی بازه‌ی نیم‌فضا پاسخ داد.*

۳.۲.۲ درخت افراز چند لایه و شمارش مثلث‌ها

برای یافتن تعداد مثلث‌های شامل نقطه‌ی پرس‌وجوی p، چهار شرط در نظر گرفته شده‌است و هر مثلثی که در آن چهار شرط صدق کند شامل نقطه‌ی داده شده خواهدبود. برای این منظور از درخت افراز چندلایه استفاده می‌کنیم. در لایه‌ی اول درخت افراز ابتدا نقاطی را که در شرط اول صدق می‌کنند، به‌دست می‌آوریم، سپس در لایه‌ی دوم به‌ازای نقاطی که در شرط اول صدق می‌کنند نقاطی را در نظر می‌گیریم که در شرط دوم صدق کنند. به‌همین ترتیب برای نقاط به‌دست آمده در این مرحله، نقاطی را انتخاب می‌کنیم که در شرط سوم صدق کنند و در مرحله‌ی آخر از بین نقاطی که در این سه شرط صدق می‌کنند نقاطی را که در شرط چهارم نیز صدق می‌کنند، به‌دست می‌آوریم. به این ترتیب تعداد مثلث‌ها را می‌توان به‌دست آورد و همچنین آنها را گزارش کرد. دقت شود که اگر بخواهیم مثلث‌ها را گزارش کنیم به اندازه‌ی تعداد مثلث‌ها به زمان پرس‌وجو اضافه می‌شود. داده‌ساختار درخت افراز

برای پاسخگویی به سوالات مشابهی نیز به کار می‌رود. به‌عنوان مثال ورودی مجموعه‌ای از پاره‌خط‌ها و هدف بدست آوردن تعداد پاره‌خط‌هایی است که توسط خط پرس و جوی دلخواهی قطع می‌شود، است. یا به ازای یک نیم‌خط داده شده بدست آوردن اولین پاره‌خطی که توسط این نیم‌خط قطع می‌شود. تمامی این مسایل مرتبط هستند و کاربردهای بسیاری در هندسه‌ی محاسباتی دارند. برای اطلاعات بیشتر علاوه بر مراجع اشاره شده می‌توان به [۳، ۵، ۷، ۲۴، ۳۹] مراجعه کرد.

داده‌ساختارهای مطرح شده در فصل‌های بعدی تحقیق مورد استفاده قرارخواهند گرفت. در ادامه الگوریتم‌های جدید برای حل مسایل عنوان شده را بیان می‌کنیم.

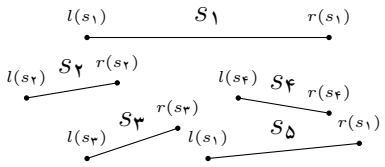
فصل ۳

الگوریتم تصادفی اول برای مسأله‌ی شمارش

در این فصل الگوریتمی تصادفی برای به‌دست آوردن جواب تقریبی مسأله‌ی شمارش ارائه می‌شود. ابتدا برای هر نقطه‌ی پرس‌وجو یک گراف نسبت می‌دهیم. سپس ثابت می‌کنیم که این گراف مسطح است و تعداد وجه‌های این گراف منهای یک یا دو مساوی تعداد پاره‌خط‌هایی است که دیده نمی‌شوند. از فرمول اویلر برای گراف‌های مسطح استفاده می‌کنیم و الگوریتمی تقریبی برای محاسبه‌ی تعداد وجه‌ها ارائه می‌دهیم. همچنین ضریب تقریب و زمان اجرای الگوریتم را محاسبه می‌کنیم. در این الگوریتم، ضریب تقریب و زمان اجرا را در مقایسه با روش‌های پیشین بهبود داده‌ایم.

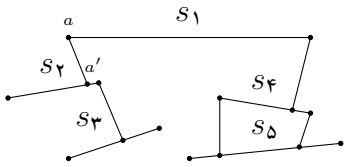
۱.۳ گراف نقطه‌ی پرس‌وجو و نتیجه‌ی اصلی

به‌ازای هر نقطه‌ی $a' \in s_i$، شعاع ساطع شده از p به a' را با $pa' \rightarrow$ نشان می‌دهیم. نقطه‌ی $a = pr(a')$ را محل برخورد $pa' \rightarrow$ و اولین پاره‌خط در S یا مربع B در نظر بگیرید. در این حالت می‌گوییم $a = pr(a)$ توسط a' پوشیده شده‌است و تصویر a' نقطه‌ی a است. همچنین فرض کنید که $\overline{x'y'}$ قسمتی از s_i و $\overline{xy}$ قسمتی از s_j است به‌طوری که $pr(x') = x$ و $pr(y') = y$ و به ازای هر نقطه‌ی $z' \in \overline{x'y'}$ داریم $pr(z') \in \overline{xy}$. در این‌صورت می‌گوییم $\overline{xy}$ توسط $\overline{x'y'}$ پوشیده شده‌است. برای هر نقطه‌ی p گرافی با نام $G(p)$ به این‌صورت می‌سازیم که به‌ازای هر پاره‌خط $s_i \in S$ راس v_i را در نظر می‌گیریم. به‌ازای هر دو راس v_i و v_j اگر s_j یک نقطه‌ی انتهایی s_i را بپوشاند (و یا برعکس، s_i یک نقطه‌ی انتهایی s_jرا بپوشاند) بین v_i و v_j یک یال قرار می‌دهیم. و در صورتی‌که هر دو نقطه‌ی انتهایی s_i توسط s_j پوشیده شوند (و یا برعکس، هر دو نقطه‌ی انتهایی s_j توسط s_i پوشیده شود) دو یال بین v_i و v_j قرار می‌دهیم (شکل ۱.۳).



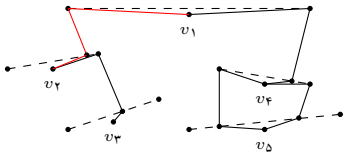
p.

(الف)



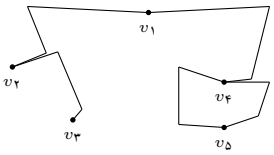
p.

(ب)



p.

(ج)



p.

(د)

شکل ۱.۳: مراحل رسم نشاندن مسطح گراف $G(p)$، (الف) پاره‌خط‌های $s_1, \ldots, s_5$ و نقاط انتهایی سمت راست و چپ پاره‌خط‌ها نشان داده شده‌اند. (ب) برای هر نقطه‌ی انتهایی $a \in s_i$ که توسط p دیدپذیر نیست، اگر $a' \in s_j$ وجود داشته باشد طوری‌که $pr(a') = a$، خط $\overline{aa'}$ را رسم می‌کنیم. (ج) برای هر پاره‌خط s_i، راس v_i را در فاصله‌ی خیلی کوتاه از وسط s_i قرار می‌دهیم. برای هر a و a' (که در (ب) توضیح داده شد)، a را به v_i و a' را به v_j وصل می‌کنیم. به این ترتیب یالی بین v_i و v_j ایجاد می‌شود. (د) پاره‌خط‌های اولیه را حذف می‌کنیم. چیزی که باقی می‌ماند گراف مسطح $G(p)$ است. توجه کنید که ۵ راس و ۵ یال داریم و هر یالی به‌صورت ۳ پاره‌خط که به‌طور متوالی به‌هم وصلند، رسم شده است.

به‌ازای هر پاره‌خط $s \in S$ نقطه‌های $l(s)$ و $r(s)$ اولین و دومین نقطه‌ی انتهایی s هستند که توسط شعاعی که حول p به صورت ساعت‌گرد حرکت می‌کند جاروب می‌شوند (شکل ۱.۳.(a)).

لم ۱.۳ *$G(p)$ را می‌توان به صورت مسطح در صفحه رسم کرد.*

اثبات. برای هر نقطه‌ی انتهایی $a \in s_i$ که توسط p دید نمی‌شود، فرض کنید داریم $a' \in s_j$ طوری‌که $pr(a') = a$. خط راست $\overline{aa'}$ را رسم می‌کنیم. در این‌صورت مجموعه‌ای از پاره‌خط‌هایی خواهیم داشت که هیچ دو تایی همدیگر را قطع نمی‌کنند. به‌ازای هر پاره‌خط s_i، راس v_i را در نقطه‌ای نزدیک به نقطه‌ی میانی پاره‌خط s_i قرار می‌دهیم. همچنین برای هر پاره‌خط $\overline{aa'}$ نقطه‌ی a را به v_i و a' را به v_j وصل می‌کنیم. در این حالت یالی شامل ۳ پاره‌خط راست $\overline{v_ia}$، $\overline{aa'}$ و $\overline{a'v_j}$ ایجاد می‌شود که v_i را به v_j وصل می‌کند. واضح است که هیچ‌کدام از این یال‌ها همدیگر را قطع نمی‌کنند. در پایان تمامی پاره‌خط‌های اولیه را حذف می‌کنیم. باقیمانده‌ی پاره‌خط‌ها و نقطه‌ها، گراف مسطح $G(p)$ را می‌سازند. این مراحل در شکل ۱.۳ آمده‌است. ■

در ادامه‌ی این بخش $G(p)$ را به عنوان گراف مسطح $G(p)$ در نظر می‌گیریم. می‌دانیم که فرمول اویلر برای گراف‌های مسطح به این شکل است [۱۷]:

$$V(G) - E(G) + F(G) = ۱ + C(G).$$

که $E(G)$ ، $V(G)$، $F(G)$ و $C(G)$ به ترتیب تعداد یال‌ها، راس‌ها، وجه‌ها و تعداد مولفه‌های همبندی G هستند. قضیه‌ی زیر روشی را برای محاسبه‌ی m_p توسط $G(p)$ ارائه می‌دهد.

قضیه ۲.۳ *تعداد پاره‌خط‌هایی که توسط p دیده نمی‌شوند، در صورتی‌که p داخل یکی از وجه‌های درونی $G(p)$ باشد مساوی $F(G(p)) - ۲$ و در غیر این‌صورت مساوی $F(G(p)) - ۱$ خواهدبود.*

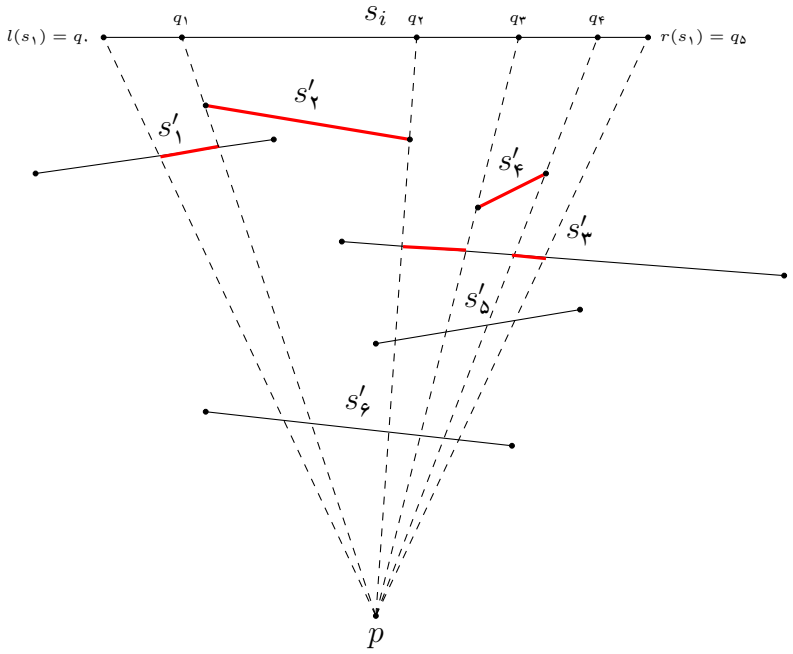
اثبات. برای اثبات یک تابع یک به یک و پوشا ϕ بین پاره‌خط‌های غیردیدپذیر از p و وجه‌های $G(p)$ به جز وجه بیرونی و وجه شامل p تعریف می‌کنیم. فرض کنید که s_i پاره‌خطی است که توسط p دیده نمی‌شود. در این‌صورت می‌توانیم s_i را به k بخش $\overline{q_۰q_۱}, \overline{q_۱q_۲}, \ldots$ افراز کنیم به‌طوری که $q_۰ = l(s_i)$، $q_k = r(s_i)$ و برای هر $\overline{q_iq_{i+۱}}$ یک بخش $\overline{q'_iq'_{i+۱}} \in s_j$ وجود دارد که $\overline{q_iq_{i+۱}}$ را پوشش می‌دهد. فرض کنید $s'_۱, s'_۲, ...,$ مجموعه‌ی پاره‌خط‌هایی باشند که $\overline{xy} \in s'_{i+۱}$ بخش $\overline{q_iq_{i+۱}}$ را می‌پوشاند. توجه کنید که برخی از پاره‌خط‌ها بیش از یک بار در این سری ظاهر می‌شوند (شکل ۲.۳). ادعا می‌کنیم که راس‌های $v'_۱, v'_۲, ..., v'_k$ و v_i، یک وجه داخلی از $G(p)$ را می‌سازند که شامل p نیست. این وجه را به s_i نسبت می‌دهیم. از آن‌جایی که $v'_۱$ راسی است که به اولین پاره‌خطی نسبت داده شده‌است که $\overline{q_۰q_۱}$ را

می‌پوشاند، دراین‌صورت s'_1 نقطه‌ی $l(s_i)$ را پوشش می‌دهد. به‌طور مشابه، s'_k نقطه‌ی $r(s_i)$ را پوشش می‌دهد و بنابراین v_i با v'_1 و v'_k مجاور است. بخش بعدی که قسمتی از s_i را می‌پوشاند از s'_2 است. بنابراین $r(s'_1)$ توسط s'_2 و یا $l(s'_2)$ توسط s'_1 پوشیده می‌شود. بنابراین v'_1 و v'_2 مجاور خواهند بود. به‌طور مشابه می‌توانیم نشان دهیم که برای هر $1 \leq i < k$، v'_i و v'_{i+1} مجاور هستند. برای تکمیل اثبات باید نشان دهیم که مسیر بسته‌ی ایجاد شده توسط $v_i \to v'_1 \to v'_2, ... \to v'_k \to v_i$ شامل p نیست. شعاع ساطع شده از p را در نظر بگیرید. ناحیه‌ای که زیر s_i و بالای s'_k قرار دارد توسط راس‌های $v'_1, v'_2, ..., v'_k$ و v_i محصور شده‌است. واضح است که p داخل این ناحیه نیست و بنابراین p داخل مسیر بسته‌ی ایجاد شده نیز قرار ندارد. حال نشان می‌دهیم که تابع ϕ یک به یک و پوشا است. ابتدا یک به یک بودن را ثابت می‌کنیم. اگر $\phi(s_i) = \phi(s_j)$ در این‌صورت طبق تعریف ϕ، قسمتی از پاره‌خط s_i قسمتی از پاره‌خط s_j را می‌پوشاند و همچنین قسمتی از پاره‌خط s_j قسمتی از پاره‌خط s_i می‌پوشاند. از آنجایی که پاره‌خط‌ها یکدیگر را قطع نمی‌کنند به تناقض می‌رسیم. برای اثبات پوشا بودن باید نشان دهیم که برای هر وجه داخلی f که شامل p نیست، راس v_i متناظر با پاره‌خط s_i وجود دارد که توسط p دیده نمی‌شود و $\phi(s_i) = f$.

برای بدست آوردن s_i از نیم‌خط شعاعی حول p استفاده می‌کنیم. از آنجایی که f وجه داخلی است و شامل p نیست، وجه f بین دو شعاعی که از p ساطع می‌شوند خواهد بود. یکی از این شعاع‌ها از سمت راست و دیگری از سمت چپ است. اگر از سمت چپ به راست شروع به جاروب کنیم پاره‌خطی متناظر با یکی از راس‌های f وجود خواهد داشت که نقطه‌ی انتهایی آن اولین نقطه‌ای خواهد بود که توسط دیگر پاره‌خط‌های متناظر با راس‌های f پوشیده می‌شود. حال ادعا می‌کنیم که s_i توسط p دیده نمی‌شود و $\phi(s_i) = f$. برای مثال در شکل ۲.۳ مسیر بسته‌ی $v_i \to v'_1 \to v'_2 \to v'_3 \to v'_4, \to v'_3, \to v_i$ یکی از وجه‌های $G(p)$ را تشکیل می‌دهد و s_i اولین پاره‌خط بین $\{s_i, s'_1, s'_2, s'_3, s'_4\}$ است که $l(s_i)$ توسط یکی از پاره‌خط‌های $\{s_i, s'_1, s'_2, s'_3, s'_4\}$ پوشیده می‌شود.

واضح است که $l(s_i)$ توسط p دیده نمی‌شود. v'_1 و v_i مجاور هستند و بنابراین قسمتی از s'_1 از s_i را می‌پوشاند. از آنجایی که v'_1 و v'_2 مجاور هستند بنابر این قسمتی از s'_2 قسمتی از s_i را درست بعد از s'_1 می‌پوشاند. با ادامه‌ی این استدلال نتیجه می‌گیریم قسمتی از هر s'_i بخشی از s_i را به صورت ممتد می‌پوشاند. v'_k و v_i نیز مجاور هستند بنابراین $r(s_i)$ توسط p دیده نمی‌شود. پس نتیجه می‌گیریم که قسمت‌هایی از s_k و ... s'_1, s'_2 به‌طور کامل s_i را می‌پوشانند و درنتیجه s_i توسط p دیده نمی‌شود.

بنابراین اگر p در وجه بیرونی $G(p)$ باشد، تعداد پاره‌خط‌هایی که توسط p دیده نمی‌شوند مساوی $F(G(p)) - 1$ و در غیراین‌صورت مساوی $F(G(p)) - 2$ است. ■



شکل ۲.۳: s_i توسط p دیده نمی‌شود و می‌تواند به ۵ قسمت $\overline{q_0q_1}, \overline{q_1q_2}, \overline{q_2q_3}, \overline{q_3q_4}, \overline{q_4q_5}$ تقسیم شود طوری‌که هر بخش با پاره‌خط‌هایی از $s'_1, s'_2, s'_3, s'_4, s'_5$ پوشیده شده‌اند.

از فرمول اویلر برای محاسبه‌ی $F(G(p))$ استفاده می‌کنیم. واضح است که $V(G(p))$ مساوی n است. برای هر نقطه‌ی انتهایی که توسط p دید نمی‌شود یک یال به $G(p)$ اضافه می‌کنیم. بنابراین $E(G(p))$ مساوی $2n - ve_p$ خواهد بود (برای یاد آوری اشاره می‌کنیم که ve_p تعداد نقاط انتهایی دیدپذیر است). فرمول اویلر و قضیه‌ی ۲.۳ لم زیر را نتیجه می‌دهد.

لم ۳.۳ *اگر p داخل یک وجه درونی $G(p)$ باشد در این‌صورت $m_p = ve_p - C(G(p)) + 1$ و در غیراین‌صورت $m_p = ve_p - C(G(p))$.*

در ادامه‌ی این فصل ۲ الگوریتم یکی برای تقریب مقدار ve_p و دیگری برای تقریب $C(G(p))$ بیان می‌کنیم و یک مقدار تقریبی از m_p را محاسبه می‌کنیم. نتیجه‌ی اصلی این بخش را در قضیه‌ی زیر بیان و در پایان فصل این قضیه را ثابت می‌کنیم.

قضیه ۴.۳ *الگوریتمی وجود دارد که به ازای هر $0 < \delta \leq 1$ و $0 \leq \beta \leq \frac{2}{3}$ مسأله‌ی شمارش را در زمان مورد انتظار پرس‌وجوی $O(\frac{1}{\delta^3} m^{\beta/2} \log m)$، با استفاده از زمان پیش‌پردازش $O(m^{2-3\beta/2} \log m)$ و حافظه‌ی مورد انتظار $O(m^{2-3\beta/2})$ به‌طور تقریبی حل می‌کند. این الگوریتم مقدار m'_p را به عنوان خروجی برمی‌گرداند به‌طوری که اگر $m_p \geq \frac{1}{\delta^3} m^{\beta/2} \log m$، با احتمال حداقل $1 - \frac{1}{\log m}$ خواهیم داشت: $m_p \leq m'_p \leq (1+\delta) m_p$ و در صورتی‌که $m_p < \frac{1}{\delta^3} m^{\beta/2} \log m$، مقدار دقیق m_p برگردانده می‌شود.*

۲.۳ الگوریتم تقریبی برای محاسبه‌ی تعداد نقاط انتهایی دیدپذیر

در این بخش الگوریتمی برای به‌دست آوردن مقدار تقریبی ve_p ارائه می‌دهیم.

در زمان پیش‌پردازش داده‌ساختار الگوریتم معرفی شده در [۵۳] را می‌سازیم. این الگوریتم $VPS(p)$ را در زمان $O(|VPS(p)|\log(n/|VPS(p)|))$ محاسبه می‌کند که $|VPS(p)|$ تعداد راس‌های $VPS(p)$ است. الگوریتم ارائه شده در [۵۳] از یک نیم‌خط شعاعی جاروب حول p استفاده می‌کند. در طی جاروب، قسمت‌هایی از پاره‌خط‌ها که توسط p دیدپذیر هستند، محاسبه می‌شوند. در زمان پیش‌پردازش، یک پارامتر ثابت β که $0 \leq \beta \leq \frac{2}{3}$ انتخاب می‌کنیم. در زمان پرس‌وجو نیز یک پارامتر $0 < \delta \leq 1$ که مقدار ضریب تقریب الگوریتم هست را تعیین می‌کنیم. ما از الگوریتم [۵۳] برای محاسبه‌ی نقاط انتهایی دیدپذیر استفاده می‌کنیم. اگر خط جاروب قبل از شمارش $\frac{1}{\delta^3}m^{\beta/2}\log m$ از نقاط انتهایی دیدپذیر به‌طور کامل حول p جاروب شود در این‌صورت $VPS(p)$ به‌طور کامل محاسبه شده‌است و خواهیم داشت $|VPS(p)| \leq \frac{1}{\delta^3}m^{\beta/2}\log m$. در این حالت تعداد دقیق پاره‌خط‌های دیدپذیر را می‌توانیم در زمان $O(\frac{1}{\delta^3}m^{\beta/2}\log m)$ محاسبه کنیم. در غیر این‌صورت $ve_p > \frac{1}{\delta^3}m^{\beta/2}\log m$ و جواب در مرحله‌ی بعدی الگوریتم به‌دست می‌آید که در ادامه توضیح می‌دهیم.

منطقه‌ی دیدپذیر یک نقطه‌ی انتهایی a به‌صورت ستاره‌ای، شامل $m_a = O(n)$ مثلث دوبه‌دو نامتقاطع است که مثلث‌های دید a نامیده و با $VTS(a)$ نشان داده می‌شوند. توجه کنید m_a تعداد یال‌های $EVG(S)$ است که مجاور a هستند. نقطه‌ی پرس‌وجوی p توسط یک نقطه‌ی انتهایی a دیده می‌شود اگر و فقط اگر داخل یکی از مثلث‌های دید a قرار گیرد. فرض کنید VTS مجموعه‌ی مثلث‌های دید تمامی نقاط انتهایی باشد. در این‌صورت تعداد نقاط انتهایی دیدپذیر مساوی تعداد مثلث‌هایی در VTS خواهد بود که شامل p هستند. با استفاده از $EVG(S)$ می‌توانیم VTS را در زمان $O(m\log m) = O(n^2\log n)$ به‌دست آوریم و داریم $|VTS| = O(m) = O(n^2)$ [۳۵].

لم ۵.۳ *فرض کنید Δ مجموعه‌ای از n مثلث باشد. داده‌ساختاری با اندازه‌ی $O(n^2)$ وجود دارد که با زمان پیش‌پردازش $O(n^2\log n)$ تعداد مثلث‌های شامل نقطه‌ی پرس‌وجوی p را در زمان $O(\log n)$ بدست می‌آورد.*

۱.۲.۳ الگوریتم

در این بخش الگوریتم محاسبه‌ی تقریبی ve_p را در حالتی‌که در قسمت قبل به جواب نرسیدیم، بیان می‌کنیم. در این حالت داریم $m_p > \frac{1}{\delta^3}m^{\beta/2}\log m$. در زمان پیش‌پردازش زیرمجموعه‌ی تصادفی

$RVT_۱ \subset VT_S$ را انتخاب می‌کنیم به‌طوری‌که هر کدام از اعضای VT_S با احتمال $\frac{۱}{m^\beta}$ انتخاب می‌شوند.

لم ۶.۳ $E(|RVT_۱|) = O(m^{۱-\beta})$.

اثبات. فرض کنید $VT_S = \{\Delta_۱, \Delta_۲, ...\Delta_{m'}\}$ که $m' = O(m) = O(n^۲)$ و $X_i = ۱$ اگر $\Delta_i \in RTV_۱$ و در غیراین‌صورت $X_i = ۰$. خواهیم داشت:

$$E(|RVT_۱|) = E(\sum_{i=۱}^{m'} X_i) = \sum_{i=۱}^{m'} E(X_i) = \sum_{i=۱}^{m'} \frac{۱}{m^\beta} = \frac{m'}{m^\beta} = O(m^{۱-\beta}).$$

■

در زمان پیش‌پردازش تعداد $m^{\beta/۲}$ زیرمجموعه‌ی تصادفی $RVT_۱, ..., RVT_{m^{\beta/۲}}$ از VT_S را انتخاب می‌کنیم. با استفاده از لم ۵.۳، برای هر نقطه‌ی پرس‌وجوی داده شده‌ی p، تعداد مثلث‌های شامل p در هر RVT_i که با $(ve_p)_i$ نشان داده می‌شود در زمان پرس‌وجوی $O(\log m)$ با زمان پیش‌پردازش $O(m^{۲-۲\beta} \log m)$ و حافظه‌ی $O(m^{۲-۲\beta})$ محاسبه می‌شود. سپس مقدار $ve'_p = m^\beta \frac{\sum_{i=۱}^{m^{\beta/۲}} (ve_p)_i}{m^{\beta/۲}}$ به‌عنوان مقدار تقریبی ve_p برگردانده می‌شود.

۲.۲.۳ تحلیل ضریب تقریب

در این بخش ضریب تقریب الگوریتم را محاسبه می‌کنیم. قرار دهید $X_i = m^\beta (ve_p)_i$، در این‌صورت لم زیر را خواهیم داشت.

لم ۷.۳ $E(X_i) = ve_p$.

اثبات. فرض کنید $VT(p) = \{\Delta'_۱, \Delta'_۲, ..., \Delta'_{ve_p}\} \subset VT_S$ مجموعه‌ی مثلث‌هایی باشد که شامل p هستند. قرار دهید $Y_j = ۱$ اگر $\Delta'_j \in RVT_i$ و درغیراین‌صورت $Y_j = ۰$. بنابراین $(ve_p)_i = \sum_{j=۱}^{ve_p} Y_j$ و $E((ve_p)_i) = E(\sum_{j=۱}^{ve_p} Y_j) = \frac{ve_p}{m^\beta}$.

$$E(X_i) = E(m^\beta (ve_p)_i) = m^\beta E((ve_p)_i) = m^\beta \frac{ve_p}{m^\beta} = ve_p.$$

■

همچنین لم زیر نیز نتیجه می‌شود.

لم ۸.۳ $E(\frac{\sum_{i=۱}^{m^{\beta/۲}} X_i}{m^{\beta/۲}}) = ve_p$.

بنابراین $X_۱, X_۲, ..., X_{m^{\beta/۲}}$ متغیرهای تصادفی هستند که $E(X_i) = ve_p$. طبق لم چبیشف[۱] داریم:

لم ۹.۳ *فرض کنید که $X_۱, X_۲, ..., X_n$ یک سری از متغیرهای تصادفی مستقل با توزیع یکسان و با امید ریاضی متناهی $E(X_۱) = E(X_۲) = ... = \mu$ باشند، در این‌صورت خواهیم داشت:*

$$P(|\frac{X_۱+...+X_n}{n} - \mu| > \varepsilon_۱) \leq \frac{Var(X)}{n{\varepsilon_۱}^۲}.$$

لم ۱۰.۳ *با احتمال حداقل $۱ - \frac{۱}{\log m}$ داریم:*

$$(۱ - \delta)ve_p \leq ve'_p \leq (۱ + \delta)ve_p.$$

اثبات. با استفاده از لم ۹.۳، قرار می‌دهیم $\varepsilon_۱ = \delta ve_p$. در این‌جا δ مقدار ضریب تقریب الگوریتم را تعیین می‌کند. واضح است که $Var(X_i) = m^{۲\beta}(ve_p)(۱ - \frac{۱}{m^\beta})\frac{۱}{m^\beta}$. پس خواهیم داشت:

$$\mathbb{P} = P(|ve'_p - ve_p| > \delta ve_p) \leq \frac{m^\beta ve_p}{m^{\beta/۲}\delta^۲(ve_p)^۲}.$$

می‌دانیم که $ve_p \geq \frac{۱}{\delta^۳}m^{\beta/۲}\log m$، پس

$$\mathbb{P} = P(|ve'_p - ve_p| > \delta ve_p) \leq \frac{\delta}{\log m} \leq \frac{۱}{\log m}.$$

با احتمال حداقل $۱ - \mathbb{P}$ داریم:

$$(۱ - \delta)ve_p \leq ve'_p \leq (۱ + \delta)ve_p.$$

همچنین برای مقادیر m به اندازه‌ی کافی بزرگ، خواهیم داشت: $\mathbb{P} \sim ۰$. ■

۳.۲.۳ تحلیل زمان و حافظه

در مرحله‌ی اول پرس‌وجو، الگوریتم [۵۳] را اجرا می‌کنیم. زمان پیش‌پردازش و حافظه‌ی مورد استفاده در این الگوریتم $O(m\log m)$ و $O(m)$ هستند. که برای یک نقطه‌ی پرس‌وجوی p، $VP_S(p)$ را در زمان $O(|VP_S(p)|\log(n/|VP_S(p)|))$ محاسبه می‌کند. از آن‌جایی که ما این الگوریتم را حداکثر به تعداد $\frac{۱}{\delta^۳}m^{\beta/۲}\log m$ مرحله اجرا می‌کنیم، زمان پرس‌وجوی مرحله‌ی اول $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ خواهدبود.

[۱] Chebyshev

طبق لم ۶.۳، $E(|RVT_i|) = O(m^{1-\beta})$. با استفاده از لم ۵.۳ امید ریاضی زمان پیش‌پردازش و حافظه برای هر RVT_i مساوی $O(m^{2-2\beta}\log m)$ و $O(m^{2-2\beta})$ خواهدبود به‌طوری‌که در زمان $O(\log m)$ می‌توانیم $(ve_p)_i$ را محاسبه کنیم. بنابراین امید ریاضی زمان و حافظه‌ی مورد نیاز مساوی $m^{\beta/2}O(m^{2-2\beta}\log m) = O(m^{2-\frac{3}{2}\beta}\log m)$ و $m^{\beta/2}O(m^{2-2\beta}) = O(m^{2-\frac{3}{2}\beta})$ خواهد بود.

در مرحله‌ی دوم الگوریتم برای هر RVT_i مقدار $(ve_p)_i$ در زمان $O(\log m)$ محاسبه می‌شود. بنابراین زمان پرس‌وجو مساوی $O(\frac{1}{\delta^3}m^{\beta/2}\log m) + O(m^{\beta/2}\log m)$ خواهد بود و لم زیر نتیجه می‌شود.

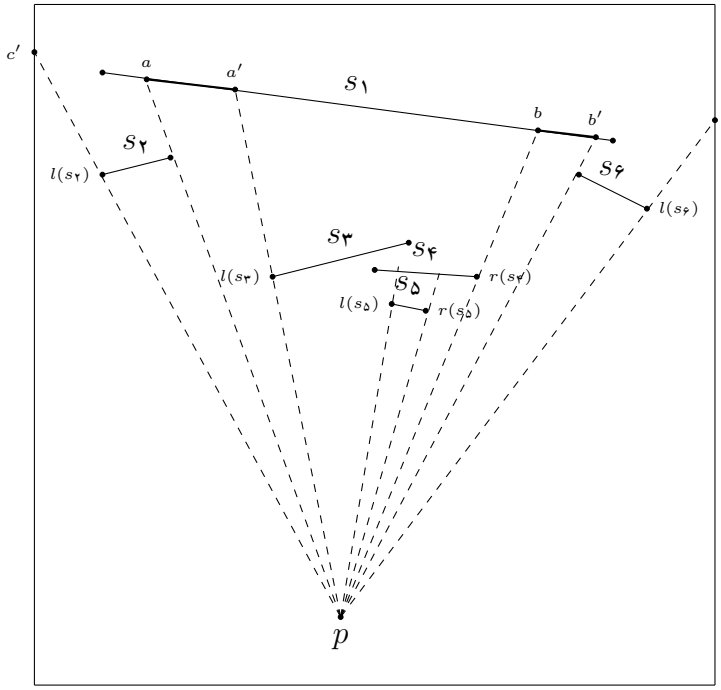
لم ۱۱.۳ *الگوریتمی وجود دارد که برای هر نقطه‌ی پرس‌وجوی p ، مقدار ve_p را در زمان پرس‌وجوی $O(\frac{1}{\delta^3}m^{\beta/2}\log m)$ تقریب می‌زند. امید ریاضی زمان پیش‌پردازش و حافظه‌ی این الگوریتم به ترتیب $O(m^{2-3\beta/2}\log m)$ و $O(m^{2-3\beta/2})$ است و داریم $(0 \le \beta \le \frac{2}{3})$. این الگوریتم در حالتی که $ve_p < \frac{1}{\delta^3}m^{\beta/2}\log m$ ، مقدار دقیق ve_p را برمی‌گرداند و در غیر این‌صورت مقدار ve'_p را برمی‌گرداند به طوری‌که با احتمال حداقل $1 - \frac{1}{\log m}$ داریم $(1-\delta)ve_p \le ve'_p \le (1+\delta)ve_p$.*

۳.۳ الگوریتم تقریبی برای محاسبه‌ی تعداد مولفه‌های همبندی $G(p)$ و اثبات قضیه‌ی ۴.۳

در این بخش الگوریتمی برای محاسبه‌ی تعداد مولفه‌های همبندی $G(p)$ ارائه می‌کنیم. فرض کنید c یک مولفه باشد که p داخل هیچ‌کدام از وجه‌های آن قرار ندارد. به راحتی می‌توان دید که شعاعی وجود دارد که از p ساطع می‌شود و هیچ‌کدام از پاره‌خط‌های متناظر با راس‌های c را قطع نمی‌کند. شروع به جاروب این شعاع در جهت ساعتگرد می‌کنیم. فرض کنید $l(c)$ (نقطه انتهایی سمت چپ c) اولین نقطه انتهایی از پاره‌خطی از c باشد که توسط این شعاع قطع می‌شود و $r(c)$ (نقطه‌ی انتهایی راست c) آخرین نقطه انتهایی باشد (شکل ۳.۳). همان‌طوری که قبلا هم گفته شد مربع شامل همه‌ی پاره‌خط‌ها، B، بخشی از $G(p)$ نیست ولی $G(p)$ داخل آن قرار دارد.

لم ۱۲.۳ *برای هر مولفه‌ی c به جز مولفه‌ای که شامل p است تصویر $l(c)$ و $r(c)$ یا هر دو به یک پاره‌خط تعلق دارند و یا هر دو متعلق به B هستند.*

اثبات. فرض کنید که $pr(l(c))$ متعلق به پاره‌خط $s \in S$ باشد. از آن‌جایی که $l(s)$ سمت چپ $l(c)$ قرار دارد، s نمی‌تواند بین پاره‌خط‌های متعلق به c باشد. ادعا می‌کنیم که $r(s)$ سمت راست $r(c)$ است. واضح است که اگر این ادعا درست باشد دراین‌صورت اگر $pr(r(c)) \in s'$، $l(s')$ سمت چپ



شکل ۳.۳: $\overline{aa'}$و $\overline{bb'}$ بخش‌های دیدپذیر از پاره‌خط s_1 هستند. B نیز یک بخش دیدپذیر از c تا c' دارد. $G(p)$ دارای ۳ مولفه‌ی $\{s_1, s_2, s_6\}$، $\{s_3, s_4\}$ و $\{s_5\}$ است. $l(s_2)$، $l(s_3)$ و $l(s_5)$ به ترتیب نقاط انتهایی چپ این پاره‌خط‌ها و $r(s_6)$، $r(s_4)$ و $r(s_5)$ هم به ترتیب نقاط انتهایی سمت راست این پاره‌خط‌ها هستند.

$l(c)$ خواهد بود. اگر $s \neq s'$ این دو پاره‌خط باید همدیگر را قطع کنند که امکان‌پذیر نیست. همچنین این نتیجه می‌دهد که اگر $pr(l(c))$ بر روی B باشد $pr(r(c))$ نیز باید بر روی B قرار گیرد. اثبات ادعا با برهان خلف خواهد بود. فرض کنید که $r(s)$ سمت چپ $r(c)$ باشد. از آن‌جایی که $r(s)$ توسط p دیدپذیر نیست در این‌صورت باید پاره‌خطی مثل s' وجود داشته باشد که $r(s)$ را می‌پوشاند. از آن‌جایی که s در c قرار ندارد و s' و s نیز به هم وصلند، s' نمی‌تواند در c باشد. بنابراین $l(s')$ سمت راست $l(c)$ قرار دارد و دیدپذیر نیست. پس باید پاره‌خط s'' وجود داشته باشد که $l(c)$ را بپوشاند. به‌طور مشابه s'' نیز نمی‌تواند در c باشد و $l(s'')$ باید توسط پاره‌خط دیگری پوشش داده شود. این فرایند نمی‌تواند به‌طور نامحدود ادامه پیدا کند (چون تعداد پاره‌خط‌ها محدود است) و بنابراین به تناقض می‌رسیم. ■

فرض کنید s'_1, s'_2, s'_3 و s'_4 ضلع‌های تشکیل دهنده‌ی B هستند. طبق لم ۱۲.۳ می‌توانیم برای هر مولفه همبندی $G(p)$ یک جفت از بخش‌های دیدپذیر مجاور یک پاره‌خط و یا یک بخش دیدپذیر از B را نسبت دهیم. برای مثال در شکل ۳.۳ ، s_1 دارای ۲ بخش دیدپذیر است که به مولفه‌ای نسبت داده شده‌است که از s_3 و s_4 تشکیل شده‌است. اگر بتوانیم تعداد بخش‌های دیدپذیر هر پاره‌خط و تعداد بخش‌های دیدپذیر B را به‌دست آوریم، در این‌صورت می‌توانیم مقدار دقیق $C(G(p))$ را

محاسبه کنیم. چون هر جفت مجاور از بخش‌های دیدپذیر هر پاره‌خط و هر بخش دیدپذیر B به یک مولفه نسبت داده می‌شود. فرض کنید c' تعداد بخش‌های دیدپذیر مربع شامل همه‌ی پاره‌خط‌ها باشد. اگر $c' > ۰$، در این‌صورت p در وجه بیرونی قرار دارد و بنابراین اگر هر پاره‌خط c_i، s_i بخش دیدپذیر داشته باشد در این‌صورت $C(G(p)) = c' + \sum_{i=۱}^{n} \max\{(c_i - ۱), ۰\}$ خواهد بود. برای مثال در شکل ۳.۳ داریم: $c_۱ = ۲$، $c_۲ = ۱$، $c_۳ = ۱$، $c_۴ = ۲$، $c_۵ = ۱$ و $c_۶ = ۱$ و همچنین $c' = ۱$ که نتیجه می‌دهد $C(G(p)) = ۳$. اگر $c' = ۰$ در این‌صورت p داخل یک وجه درونی است و این وجه در مولفه‌ای قرار دارد که نقطه انتهایی چپ و راست ندارد و بنابراین در این حالت داریم $C(G(p)) = ۱ + \sum_{i=۱}^{n} \max\{(c_i - ۱), ۰\}$.

در ادامه الگوریتمی برای تقریب تعداد بخش‌های دیدپذیر هر پاره‌خط $s_i \in S \cup \{s'_۱, s'_۲, s'_۳, s'_۴\}$ ارائه می‌دهیم.

۱.۳.۳ الگوریتم

طبق [۳۵]، منطقه دیدپذیر توسط هر پاره‌خط $s_i \in S \cup \{s'_۱, s'_۲, s'_۳, s'_۴\}$ را می‌توانیم توسط $O(m_{s_i})$ مثلث که با $VT(s_i)$ نشان داده می‌شوند پوشش دهیم. در اینجا $|VT(s_i)| = O(m_{s_i})$ که m_{s_i} تعداد یال‌های $EVG(S)$ است که مجاور با s_i هستند. توجه کنید که مثلث‌های دید s_i ممکن است همدیگر را قطع کنند. اگر مثلث‌های دید تمامی پاره‌خط‌ها را در نظر بگیریم در این‌صورت مجموعه‌ی $VT_S = \{\Delta_۱, \Delta_۲, ...\}$ از $|VT_S| = O(m)$ مثلث خواهیم داشت.

می‌گوییم Δ_i مرتبط با s_j است اگر و فقط اگر $\Delta_i \in VT(s_j)$. برای یک نقطه‌ی پرس‌وجوی داده‌شده‌ی p ، m''_p، تعداد مثلث‌های VT_S که شامل p هستند بین m_p و $۲m_p$ قرار دارد و بنابراین m''_p یک جواب با ضریب تقریب ۲ برای مسأله‌ی شمارش به‌دست می‌دهد [۳۵]. از آن‌جایی که مثلث‌های دید یک پاره‌خط ممکن است همدیگر را قطع کنند، تعدادی از پاره‌خط‌ها چندین بار شمرده می‌شوند. در [۳۵] نشان داده‌شده‌است که هر پاره‌خط s_i، c_i مرتبه شمرده می‌شود که c_i تعداد بخش‌های دیدپذیر s_i است. به‌عبارت دیگر c_i مثلث مرتبط با s_i در VT_S وجود دارند که شامل p هستند. روشی مشابه بخش قبل برای تقریب $C(G(p))$ استفاده می‌کنیم. یک زیر مجموعه‌ی تصادفی $RVT_۱ \subset VT_S$ انتخاب می‌کنیم. به‌طوری که هر عضو VT_S با احتمال $\frac{۱}{m^\beta}$ انتخاب می‌شود. برای یک نقطه‌ی پرس‌وجوی داده شده فرض کنید $c'_{i,۱} \geq ۱$ تعداد مثلث‌های متناظر با s_i در $RVT_۱$ باشد که شامل p هستند. مقدار $C_۱ = \sum_{i=۱}^{n}(m^\beta c'_{i,۱} - ۱)$ را به‌عنوان مقدار تقریبی $C(G(p))$ به‌دست آمده از $RVT_۱$ گزارش می‌کنیم. تعداد $m^{\beta/۲}$ زیرمجموعه‌ی تصادفی $RVT_۱, ..., RVT_{m^{\beta/۲}}$ از VT_S را انتخاب

می‌کنیم. فرض کنید p نقطه ی پرس و جوی داده شده باشد. برای هر RVT_j، $C_j = \sum_{i=۱}^{n}(m^{\beta}c'_{i,j}-۱)$ را محاسبه می‌کنیم. در پایان $C'_p = \frac{\sum_{j=۱}^{m^{\beta/۲}} C_j}{m^{\beta/۲}}$ به‌عنوان مقدار تقریبی $C(G(p))$ گزارش می‌شود.

۲.۳.۳ تحلیل ضریب تقریب

نشان می‌دهیم که اگر $C(G(p)) > \frac{۱}{\delta^۲}m^{\beta/۲}\log m$ با احتمال حداقل $\frac{۱}{\log m}$ ، C'_p یک مقدار تقریبی از $C(G(p))$ با ضریب تقریب $(۱+\delta)$ است.

لم ۱۳.۳ $E(C_j) = C(G(p))$.

اثبات. $E(C_j) = E(\sum_{i=۱}^{n} m^{\beta}c'_{i,j} - ۱) = \sum_{i=۱}^{n} E(m^{\beta}c'_{i,j} - ۱) =$
$\sum_{i=۱}^{n} c_i - ۱ = C(G(p)).$ ■

با استفاده از لم ۹.۳ خواهیم داشت:

$$\mathbb{P} = P(|\frac{C_۱+\ldots+C_{m^{\beta/۲}}}{m^{\beta/۲}} - C(G(p))| > \delta C(G(p))) \leq \frac{Var(C_i)}{m^{\beta/۲}\delta^۲ C(G(p))^۲}.$$

$Var(C_i) = m^{۲\beta}C(G(p))(\frac{۱}{m^{\beta}})(۱-\frac{۱}{m^{\beta}})$. از آنجایی که $C(G(p)) > \frac{۱}{\delta^۲}m^{\beta/۲}\log m$ بنابراین

$$\mathbb{P} = P(|\frac{C_۱+\ldots+C_{m^{\beta/۲}}}{m^{\beta/۲}} - C(G(p))| > \delta C(G(p))) \leq \frac{۱}{\log m}.$$

پس با احتمال حداقل $\mathbb{P}-۱$ داریم:

$$(۱-\delta)C(G(p)) \leq C'_p \leq (۱+\delta)C(G(p)).$$

و برای مقادیر بزرگ m داریم: $\mathbb{P} \sim ۰$.

۳.۳.۳ تحلیل زمان و حافظه‌ی مورد نیاز

با استفاده از لم ۵.۳ برای هر RVT_i داده‌ساختاری با زمان پیش‌پردازش و حافظه‌ی مورد انتظار $O(m^{۲-۲\beta}\log m)$ و $O(m^{۲-۲\beta})$ نیاز داریم. RVT_i مقدار C_i را در زمان $O(\log m)$ برای هر نقطه‌ی پرس‌وجوی p برمی‌گرداند. بنابر این امید ریاضی حافظه‌ی مورد نیاز برای تمامی $m^{\beta/۲}$ داده‌ساختار مساوی $O(m^{۲-۲\beta+\beta/۲}\log m)$ و امید ریاضی زمان پرس‌وجوی محاسبه‌ی C'_p مساوی $O(m^{\beta/۲}\log m)$ می‌شود. از این رو داریم:

لم ۱۴.۳ *$C(G(p))$ را می‌توانیم در زمان $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ تقریب بزنیم. امید ریاضی زمان پیش‌پردازش و حافظه‌ی مورد نیاز مساوی $O(m^{۲-۳\beta/۲})$ و $O(m^{۲-۳\beta/۲})$ خواهد بود. برای هر نقطه‌ی پرس‌وجوی p، وقتی $C(G(p)) > \frac{۱}{\delta^۳}m^{\beta/۲}\log m$ این الگوریتم مقدار C'_p را با احتمال حداقل $۱ - \frac{۱}{\log m}$ طوری برمی گرداند که $(۱-\delta)C(G(p)) \le C'_p \le (۱+\delta)C(G(p))$.*

۴.۳.۳ اثبات قضیه ۴.۳

فرض کنید که داده‌ساختارهای [۵۳] و الگوریتم‌های لم‌های ۱۱.۳ و ۱۴.۳ را داریم. برای نقطه‌ی پرس‌وجوی p، در ابتدا الگوریتم [۵۳] را $\frac{۱}{\delta^۳}m^{\beta/۲}\log m$ مرحله اجرا می‌کنیم. اگر $VP_S(p)$ محاسبه شود در این‌صورت مقدار دقیق m_p در زمان $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ محاسبه می‌شود. در غیر این‌صورت مقدار ve'_p و C'_p را در زمان $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ محاسبه می‌کنیم در مرحله دوم خواهیم داشت $m_p > \frac{۱}{\delta^۳}m^{\beta/۲}\log m$. از آن‌جایی که $m_p \le ve_p$ خواهیم داشت: $ve_p > \frac{۱}{\delta^۳}m^{\beta/۲}$. لم ۱۱.۳ نیز نتیجه می‌دهد: $|ve'_p - ve_p| \le \delta ve_p$.

حال نشان می‌دهیم که اگر $C'_p < (۱+\delta)\frac{۱}{\delta^۳}m^{\beta/۲}\log m$، در این‌صورت با احتمال حداقل $۱-\frac{۱}{\log m}$ داریم: $C(G(p)) < \frac{۱+\delta}{۱-\delta}\frac{۱}{\delta^۳}m^{\beta/۲}\log m$. با استفاده از لم ۹.۳ داریم:

$$P(|C(G(p) - C'_p| > \delta C(G(p)) \le \frac{m^{\beta/۲}}{\delta^۲ C(G(p))}$$
$$P(C(G(p)) > \frac{C'_p}{۱-\delta}) \le \frac{m^{\beta/۲}}{\delta^۲ C(G(p))}.$$

که نتیجه می‌دهد اگر $C(G(p)) > \frac{۱}{\delta^۳}m^{\beta/۲}\log m$ در این‌صورت با احتمال حداکثر $\frac{۱}{\log m}$ خواهیم داشت: $C(G(p)) > \frac{C'_p}{۱-\delta}$. پس با احتمال حداقل $۱ - \frac{۱}{\log m}$ داریم:

$$C(G(p)) \le \frac{C'_p}{۱-\delta} \le \frac{۱+\delta}{۱-\delta}\frac{۱}{\delta^۳}m^{\beta/۲}\log m.$$

از آن‌جایی که $C(G(p)) \le m_p$ و $m_p > \frac{۱}{\delta^۳}m^{\beta/۲}\log m$، داریم: $C(G(p)) \le \frac{۱+\delta}{۱-\delta}\delta m_p$. می‌دانیم که $ve_p = m_p + C(G(p))$ و با احتمال حداقل $۱ - \frac{۱}{\log m}$ داریم: $(۱-\delta)ve_p \le ve'_p \le (۱+\delta)ve_p$. پس خواهیم داشت:

$$\frac{ve'_p}{۱+\delta} \le m_p + C(G(p)) \le m_p + \frac{۱+\delta}{۱-\delta}\delta m_p.$$

که نتیجه می‌دهد

$$m_p \le \frac{ve'_p}{۱-\delta} \le \frac{(۱+\delta^۲)(۱+\delta)}{(۱-\delta)^۲}m_p.$$

قرار دهید $۱+\delta^* = \frac{(۱+\delta^۲)(۱+\delta)}{(۱-\delta)^۲}$، پس

$$m_p \leq \frac{ve'_p}{۱-\delta} \leq (۱+\delta^*)m_p.$$

بنابراین $\frac{ve'_p}{۱-\delta}$ به عنوان مقدار تقریبی m_p گزارش می شود.

اگر $C'_p > (۱+\delta)\frac{۱}{\delta^۲}m^{\beta/۲}\log m$، طبق لم ۹.۳ داریم:

$$P(|C'_p - C(G(p)| \geq \frac{۱}{\delta}m^{\beta/۲}\log m) \leq \frac{m^{\beta/۲}C(G(p))}{\frac{۱}{\delta^۲}m^{\beta}\log^۲ m}.$$

پس $P(C(G(p)) < C'_p - \frac{۱}{\delta}m^{\beta/۲}\log m) \leq \frac{m^{\beta/۲}C(G(p))}{\frac{۱}{\delta^۲}m^{\beta}\log^۲ m}$. می‌دانیم $C'_p > (۱+\delta)\frac{۱}{\delta^۲}m^{\beta/۲}\log m$. پس اگر $C(G(p)) \leq \frac{۱}{\delta^۲}m^{\beta/۲}\log m$، داریم $P(C(G(p)) < \frac{۱}{\delta^۲}m^{\beta/۲}\log m) \leq \frac{۱}{\log m}$. که نتیجه می‌گیریم با احتمال حداقل $۱ - \frac{۱}{\log m}$ داریم $C(G(p)) > \frac{۱}{\delta^۲}m^{\beta/۲}\log m$، پس می‌توانیم از لم ۱۴.۳ استفاده کنیم.

$$\begin{aligned} m_p = ve_p - C(G(p)) &\leq \frac{ve'_p}{۱-\delta} - \frac{C'_p}{۱+\delta} \\ &\leq \frac{(۱+\delta)ve_p}{۱-\delta} - \frac{(۱-\delta)C(G(p))}{۱+\delta} \\ &\leq ve_p - C(G(p)) + \frac{۲(\delta)ve_p}{۱-\delta} + \frac{۲\delta C(G(p))}{۱+\delta}. \end{aligned}$$

می‌دانیم $C(G(p)) \leq m_p$ و همچنین $m_p \leq ve_p$ پس

$$\begin{aligned} &\leq m_p + \frac{۲\delta m_p}{۱-\delta} + \frac{\delta m_p}{۱+\delta} \\ &\leq (۱ + \frac{۲\delta}{۱-\delta} + \frac{\delta}{۱+\delta})m_p. \end{aligned}$$

قرار دهید $\delta^* = \frac{۲\delta}{۱-\delta} + \frac{\delta}{۱+\delta}$، بنابراین

$$m_p \leq \frac{ve'_p}{۱-\delta} - \frac{C'_p}{۱+\delta} \leq (۱+\delta^*)m_p.$$

و $\frac{ve'_p}{۱-\delta} - \frac{C'_p}{۱+\delta}$ به عنوان مقدار تقریبی m_p گزارش می‌شود. توجه کنید در صورتی‌که δ به اندازه‌ی کافی کوچک انتخاب شود می‌توان δ^* را به میزان دلخواه کوچک کرد.

زمان پرس‌وجوی الگوریتم ما $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ است که ارتباط δ^* و δ به این صورت است که وقتی δ از یک مقدار ثابت C کمتر باشد δ^* حداکثر به اندازه‌ی یک ضریب ثابت خطی از δ خواهد بود و بنابراین زمان پرس‌وجو را می توان به شکل $O(\frac{۱}{\delta^۳}m^{\beta/۲}\log m)$ نمایش داد . برای حالت $\delta > C$ چون $\delta^{-۳} < C^{-۳}$، زمان پرس و جو به شکل $O(m^{\beta/۲}\log m)$ خواهد بود.

۴.۳ نتیجه‌گیری

در این فصل الگوریتم تصادفی برای تقریب مقدار جواب مسأله‌ی شمارش بیان کردیم. در این الگوریتم به ازای هر نقطه‌ی پرس‌وجو گرافی مسطح تعریف می‌شود که تعداد وجه‌های آن گراف

معادل پاره‌خط‌های غیردیدپذیر است. با استفاده از فرمول اویلر برای گراف‌های مسطح می‌توان تعداد وجه‌ها را که وابسته با تعداد یال‌ها و راس‌ها و مولفه‌های همبندی گراف است، بدست آورد. از آنجایی که روشی قطعی برای بدست آوردن تعداد مولفه‌ها نداشتیم، روشی تصادفی برای تقریب تعداد مولفه‌ها ارائه دادیم و جواب مسأله را به‌طور تقریبی بدست آوردیم. البته ضریب تقریب و همچنین زمان اجرا و حافظه‌ی این الگوریتم در مقایسه با روش‌های پیشین را بهبود دادیم. مسأله‌ای که در آینده می‌توان بررسی کرد بدست آوردن تعداد دقیق مولفه‌های گراف بدست آمده در زمان مورد انتظار ما است.

فصل ۴

الگوریتم تصادفی دوم و یک الگوریتم دقیق برای مسأله‌ی شمارش

در این فصل الگوریتم تصادفی دیگری برای مسأله‌ی شمارش بیان می‌کنیم. اساس این الگوریتم نیز بر پایه‌ی نمونه‌گیری تصادفی است. بعد از بیان الگوریتم و تحلیل زمان اجرا، نتایج تجربی را که با استفاده از پیاده‌سازی الگوریتم بدست آمده است، بیان می‌کنیم. سپس الگوریتمی برای بدست آوردن مقدار دقیق m_p ارائه می‌شود. حافظه و زمان پرس و جوی این روش همانند روش ارائه شده در [۳۵] است. ولی زمان پیش‌پردازش بیشتر از [۳۵] است. توجه شود که در[۳۵] مقدار تقریبی m_p محاسبه می‌شود.

۱.۴ الگوریتم تصادفی

برای ارائه این الگوریتم ابتدا قضیه‌ی زیر را بیان می‌کنیم. اثبات این قضیه در [۱۱] آمده‌است.

قضیه ۱.۴ *با زمان پیش‌پردازش $O(n^۲)$، به ازای یک نقطه‌ی پرس‌وجو و یک پاره‌خط تصادفی که از بین پاره‌خط‌های مجموعه‌ی S انتخاب می‌شود می‌توان مسأله‌ی دیدپذیری را برای آن نقطه و پاره‌خط در زمان مورد انتظار $O(\log n)$ پاسخ داد.*

در ادامه با نمونه‌گیری تصادفی جواب مسأله‌ی شمارش را به طور تقریبی بدست می‌آوریم. برای بهبود زمان اجرا از قضیه‌ای که بیان کردیم استفاده می‌کنیم. مشابه فصل قبل ابتدا الگوریتم ارائه شده در [۵۳] را برای $O_\epsilon(\sqrt{n})$ مرتبه اجرا می‌کنیم. اگر قبل از شمارش $\sqrt{n}$ تعداد از پاره‌خط‌های دیدپذیر از

p، الگوریتم پایان یابد، $VP_S(p)$ به‌طور کامل محاسبه شده‌است و $|VP_S(p)| \leq \sqrt{n}$، پس می‌توانیم مقدار دقیق m_p را از $VP_S(p)$ به‌دست آوریم. در غیر این‌صورت خواهیم داشت $|VP_S(p)| > \sqrt{n}$. در این حالت ابتدا یک زیرمجموعه‌ی تصادفی $RS_1 \subset S$ را انتخاب می‌کنیم به‌طوری که هر پاره‌خط با احتمال $\frac{1}{\sqrt{n}}$ در مجموعه‌ی $RS_1 \subset S$ قرار می‌گیرد. سپس برای هر پاره‌خط $s_i \in RS_1$ با استفاده از قضیه ۱.۴ بررسی می‌کنیم که آیا s_i و p در مجموعه‌ی S دیدپذیر هستند. فرض کنید X_1 تعداد پاره‌خط‌های دیدپذیر از p در RS_1 باشد (دیدپذیری بر اساس مجموعه‌ی S بررسی می‌شود). این الگوریتم را برای t زیرمجموعه‌ی تصادفی $RS_i \subset S$ اجرا می‌کنیم. فرض کنید X_i تعداد پاره‌خط‌های دیدپذیر از p در RS_i باشد، مقدار $m'_p = \frac{\sum_{i=1}^{t} \sqrt{n} X_i}{t}$ را به‌عنوان مقدار تقریبی m_p بر می‌گردانیم. با استفاده از ۹.۳ داریم:

لم ۲.۴ *با احتمال نزدیک به ۱ خواهیم داشت:*

$$(1-\delta)m_p \leq \sqrt{n}\frac{X_1+X_2+\ldots+X_t}{t} \leq (1+\delta)m_p.$$

اثبات. در لم ۹.۳ قرار می‌دهیم $\varepsilon = \delta m_p$ و $t = \frac{1}{\delta^2}$. واضح است که $E(\sqrt{n}X_i) = m_p$ و $Var(\sqrt{n}X_i) = n(m_p)(1-\frac{1}{\sqrt{n}})\frac{1}{\sqrt{n}}$. بنابر این داریم:

$$\mathbb{P} = P(|\sqrt{n}\frac{X_1+X_2+\ldots+X_t}{t} - m_p| > \delta m_p) \leq \frac{\sqrt{n}m_p}{{m_p}^2}.$$

از آن‌جایی که $m_p > O_\epsilon(\sqrt{n})$، بنابراین $\mathbb{P} \sim 0$ که نتیجه می‌دهد با احتمال حداقل $\mathbb{P} - 1$ خواهیم داشت:

$$(1-\delta)m_p \leq m'_p \leq (1+\delta)m_p.$$

■

۲.۴ تحلیل

در مرحله‌ی اول، الگوریتم ارائه شده در [۵۳] را برای $O_\epsilon(\sqrt{n})$ مرتبه اجرا می‌کنیم. در مرحله‌ی دوم طبق قضیه‌ی ۱.۴ در زمان مورد انتظار $O(\log n)$ می‌توانیم برای هر پاره‌خط مسأله‌ی دیدپذیری را پاسخ دهیم. از آن‌جایی‌که $O(\sqrt{n})$ پاره‌خط برای هر RS_i انتخاب می‌کنیم، زمان پرس‌وجو مساوی $O(t\sqrt{n}\log n)$ خواهدبود. درنتیجه زمان پیش‌پردازش و حافظه طبق مطالب فصل قبل، $O_\epsilon(n^2)$ و $O(n^2)$ خواهد بود. بنابراین لم زیر که نتیجه‌ی اصلی این فصل است، بدست می‌آید.

قضیه ۳.۴ *با امید ریاضی زمان پیش‌پردازش و حافظه‌ی $O_\epsilon(n^۲)$ و $O(n^۲)$ می‌توان مسأله‌ی شمارش را در زمان مورد انتظار $O_\epsilon(\frac{۱}{\delta^۲}\sqrt{n})$ پاسخ داد. این الگوریتم در صورتی‌که $m_p < \sqrt{n}$ باشد مقدار دقیق جواب مسأله را بدست می آورد و در غیر این‌صورت m'_p را بر می‌گرداند به‌طوری‌که با احتمال نزدیک به ۱ خواهیم داشت:*

$$(۱ - \delta)m_p \leq m'_p \leq (۱ + \delta)m_p.$$

۳.۴ نتایج تجربی

برای بررسی زمان اجرا، الگوریتم ارائه شده را پیاده‌سازی کرده‌ایم. در این بخش نتایج تجربی به‌دست آمده را بیان می‌کنیم. در ابتدا برای مسأله‌ی دیدپذیری میانگین زمان پرس‌وجو را برای تعداد پاره‌خط‌های متفاوت و نقاط پرس‌وجوی متفاوت به‌دست آورده‌ایم. طبق قضیه‌ی ۱.۴ امید ریاضی زمان اجرا به ازای هر نقطه ثابت و یک پاره‌خط تصادفی انتخاب شده مساوی $O(\log n)$ است. نتایج تجربی نیز این قضیه را تایید می‌کنند. همچنین برای مسأله‌ی شمارش به‌ازای مقادیر مختلف n، δ و نقاط پرس و جوی تصادفی انتخاب شده، مقدار دقیق تعداد پاره‌خط‌های دیدپذیر و همچنین جواب به‌دست آمده از الگوریتم را محاسبه کرده‌ایم. نتایج در جدول ۱.۴ نشان داده‌شده‌اند. نتایج به‌دست آمده نشان می‌دهند که برای حالت $m_p > \sqrt{n}$ جواب به‌دست آمده با الگوریتم ما نتایج قضیه ۱.۴را تایید می‌کند.

m'_p	m_p	$t = \frac{۱}{\delta^۲}$	δ	n
۵۷۰۰	۵۹۷۱	۱	۱	۱۰۰۰۰
۳۲۰۰	۲۹۸۵	۱	۱	۱۰۰۰۰
۶۲۰۰	۷۰۲۱	۱	۱	۱۰۰۰۰
۳۹۰۰	۳۹۷۳	۱	۱	۱۰۰۰۰
۵۷۰۰	۴۹۶۴	۴	۰,۵	۱۰۰۰۰
۴۶۰۰	۵۰۲۹	۴	۰,۵	۱۰۰۰۰
۴۴۰۰	۴۹۸۶	۴	۰,۵	۱۰۰۰۰
۶۱۵۰	۵۰۷۱	۴	۰,۵	۱۰۰۰۰
۴۶۷۵	۴۹۹۸	۱۶	۰,۲۵	۱۰۰۰۰
۴۲۶۹۰	۳۹۶۴۱	۱	۱	۱۰۰۰۰۰
۷۰۲۰۲	۸۰۰۷۰	۱	۱	۱۰۰۰۰۰
۱۷۳۹۲	۱۹۸۵۹	۱	۱	۱۰۰۰۰۰
۴۸۳۸۲	۵۰۰۲۱	۱	۱	۱۰۰۰۰۰
۵۱۰۷۰	۴۹۸۸۲	۴	۰,۵	۱۰۰۰۰۰
۵۰۷۵۴	۴۹۹۴۰	۴	۰,۵	۱۰۰۰۰۰
۵۶۶۰۴	۵۰۰۰۱	۴	۰,۵	۱۰۰۰۰۰
۴۶۱۶۹	۴۹۹۹۰	۴	۰,۵	۱۰۰۰۰۰
۴۹۶۴۷	۴۹۹۷۷	۱۶	۰,۲۵	۱۰۰۰۰۰
۴۰۹۰۰۰	۳۹۹۴۳۹	۱	۱	۱۰۰۰۰۰۰
۵۷۲۰۰۰	۶۰۰۴۷۵	۱	۱	۱۰۰۰۰۰۰
۷۱۲۰۰۰	۶۹۹۵۶۶	۱	۱	۱۰۰۰۰۰۰
۹۰۱۰۰۰	۹۰۰۴۱۸	۱	۱	۱۰۰۰۰۰۰
۵۰۹۰۰۰	۴۹۹۶۳۲	۴	۰,۵	۱۰۰۰۰۰۰
۴۹۱۰۰۰	۴۹۹۸۳۳	۴	۰,۵	۱۰۰۰۰۰۰
۴۹۹۰۰۰	۴۹۹۴۴۴	۴	۰,۵	۱۰۰۰۰۰۰
۵۰۱۵۰۰	۵۰۰۲۴۹	۴	۰,۵	۱۰۰۰۰۰۰
۵۱۲۲۵۰	۵۰۰۳۹۵	۱۶	۰,۲۵	۱۰۰۰۰۰۰

جدول ۱.۴: مقادیر m_p و m'_p برای مقادیر مختلف n، δ و t.

۴.۴ شمارش دقیق

طبق [۳۵] مناطق دیدپذیر پاره‌خط‌های مجموعه‌ی S را می‌توان با $O(m)$ مثلث پوشاند و به ازای نقطه‌ی پرس‌وجوی داده شده p، تعداد مثلث‌های شامل p، جوابی با ضریب تقریب ۲ برای m_p بدست می‌دهد. اشاره شد که اگر به ازای هر پاره‌خط s_i، مثلث‌های دید آن پاره‌خط را با رنگ c_i رنگ کنیم در این صورت تعداد مثلث‌های شامل نقطه p با رنگ‌های متمایز جواب دقیق مسأله را بدست می‌دهد. این مسأله حالت خاصی از مسأله‌ی جستجوی عمومی بازه است. حالت ساده‌تر این مسأله به این صورت بیان می‌شود که n نقطه‌ی رنگی داده شده‌اند. به‌ازای یک خط پرس‌وجوی داده شده، هدف بدست آوردن تعداد نقاط با رنگ‌های متمایز بالای آن خط است. زمان پیش‌پردازش و حافظه‌ی مورد نیاز این الگوریتم مشابه حالت معمولی مسأله است، ولی به زمان پرس‌وجوی مسأله مقدار t نیز اضافه می‌شود که t تعداد رنگ‌های متمایز است[۳۷]. واضح است در صورتی‌که t بزرگ باشد این روش کارایی ندارد. ما نیاز به الگوریتمی داریم که زمان پرس‌وجوی آن مستقل از تعداد رنگ‌ها باشد. برای این منظور الگوریتم جدیدی ارائه می‌دهیم.

۵.۴ الگوریتم

ناحیه‌هایی که از برخورد مثلث‌های دید به‌دست می‌آید را در نظر می‌گیریم. طبق مطالب فصل ۲ می‌دانیم که به ازای همه‌ی نقاط داخل یک بخش، تعداد مثلث‌های با رنگ متمایز شامل آن نقطه یکسان است. همچنین اختلاف ۲ بخش همسایه دقیقاً یک مثلث و حداکثر ۱ رنگ است. در الگوریتمی که ارائه می‌کنیم هر کدام از مثلث‌ها را با احتمال $\frac{۱}{m^{\alpha}}$ انتخاب می‌کنیم به‌طوریکه $۰ \leq \alpha \leq ۰/۵$. اگر ناحیه‌هایی که از برخورد مثلث‌های انتخاب شده، تشکیل شده است را در نظر بگیریم و صفحه به R ناحیه تقسیم شود، خواهیم داشت.

لم ۴.۴ $E(R) = O(m^{۲-۲\alpha})$.

اثبات. امید ریاضی تعداد پاره‌خط‌های انتخاب شده مساوی $O(\frac{m}{m^{\alpha}})$ است بنابراین امید ریاضی تعداد ناحیه‌هایی که تشکیل می‌شود $O(m^{۲-۲\alpha})$ است. ■

حال این R ناحیه و $O(m)$ مثلث را در نظر بگیرید. امید ریاضی تعداد مثلث‌هایی که انتخاب نشده‌اند و یک ناحیه بخصوص از این R ناحیه را قطع می‌کنند مساوی $O(m^{\alpha})$ خواهد بود. چون با $O(m)$ مثلث کل صفحه به $O(m^{۲})$ ناحیه تقسیم می‌شود. از آنجایی که با انتخاب تصادفی مثلث‌ها

R ناحیه ساخته‌ایم، بنابراین هر ناحیه از آن R ناحیه به طور متوسط شامل $O(\frac{m^۲}{R})$ تا از $O(m^۲)$ ناحیه خواهد بود. برای تشکیل $\frac{m^۲}{R}$ ناحیه نیاز به $\frac{m}{\sqrt{R}}$ پاره‌خط داریم. پس امید ریاضی تعداد مثلث‌هایی که یک ناحیه از آن R را قطع می‌کنند مساوی $O(m^\alpha)$ خواهد بود.

در الگوریتمی که ارائه می‌دهیم در زمان پیش‌پردازش به ازای هر ناحیه‌ی R_i، یک نقطه‌ی دلخواه p_i از آن ناحیه انتخاب می‌کنیم. سپس جواب دقیق مسأله را برای p_i محاسبه و ذخیره می‌کنیم.

در مرحله‌ی پیش‌پردازش داده‌ساختار درخت افراز را بر روی تمامی یال‌های مثلث‌ها می‌سازیم. هدف ما انجام پیش‌پردازش بر روی یال‌های مثلث‌ها است به‌طوریکه به‌ازای یک پاره‌خط داده شده $\overline{pp_i}$ یال‌هایی از مثلث‌ها که توسط $\overline{pp_i}$ قطع می‌شوند را بدست آوریم.

در زمان پرس‌وجو به ازای p داده شده، ناحیه R_i شامل نقطه p را در زمان $O(\log n)$ بدست می‌آوریم. سپس با استفاده از درخت‌افرازی که داریم تعداد پاره‌خط‌هایی که $\overline{pp_i}$ را قطع می‌کنند بدست می‌آوریم. حافظه و زمان پیش‌پردازش درخت افراز $O_\epsilon(k)$ و زمان پرس‌وجو از $O(\frac{m}{\sqrt{k}}+t)$ خواهد بود. که t تعداد پاره‌خط‌هایی است که $\overline{pp_i}$ را قطع می‌کنند، همچنین داریم $m \le k \le m^۲$. مقدار $k = R$ در نظر می‌گیریم و بنابراین زمان پرس و جو مساوی $O(\frac{m}{\sqrt{R}}+t)$ خواهد بود. چون p_i و p در ناحیه R_i هستند و به طور متوسط حداکثر $\frac{m}{\sqrt{R}}$ از مثلث‌ها R_i را قطع می‌کنند پس زمان پرس‌و جو مساوی $O(\frac{m}{\sqrt{R}})$ خواهد بود.

جواب دقیق مسأله برای p_i در زمان پیش‌پردازش محاسبه شده‌است. تفاوت بین p_i و p توسط مثلث‌هایی که $\overline{pp_i}$ را قطع می‌کنند مشخص می‌شود. حال پاره‌خط‌های متمایزی که پاره خط $\overline{pp_i}$ را قطع می‌کنند بدست می‌آوریم. به ازای هر رنگ با استفاده از قضیه ۱.۴ تعیین می‌کنیم که آیا p_i و s_i و p و s_i دیدپذیر هستند یا نه. اگر s_i توسط p_i و p دیدپذیر بود و یا s_i توسط p_i و p دیدپذیر نبود کاری انجام نمی‌دهیم. اگر s_i توسط p دیدپذیر بود و توسط p_i دیدپذیر نبود به مقدار جواب p_i یکی را اضافه می‌کنیم. اگر s_i توسط p_i دیدپذیر بود و توسط p دیدپذیر نبود از مقدار جواب p_i یکی کم می‌کنیم. نهایتا مقدار بدست آمده را به عنوان جواب گزارش می‌کنیم. واضح است که در این شمارش هر پاره‌خط دیدپذیری فقط یک بار شمارش می‌شو بنابراین جواب دقیق مسأله بدست می‌آید.

۶.۴ تحلیل حافظه و زمان

حافظه‌ی الگوریتم از $O(R)$ خواهد بود چون $O(R)$ حافظه برای نگهداری R ناحیه نیاز داریم و حافظه‌ی مورد نیاز قضیه ۵.۲ نیز $O(R)$ است . برای الگوریتم قضیه ۱.۴ نیز $O(n^۲)$ حافظه نیاز داریم. زمان پرس‌وجو نیز مساوی $O(\frac{m}{\sqrt{R}})$ می شود چون در زمان $O(\log n)$ ناحیه R_i را پیدا می‌کنیم

و زمان پرس و جوی قضیه ۵.۲ مساوی $O(\frac{m}{\sqrt{R}})$ است. برای هر کدام از $O(\frac{m}{\sqrt{R}})$ از پاره‌خط‌ها نیز $O(\log n)$ زمان برای تعیین دیدپذیری p و p_i و s_i صرف می‌کنیم. پس کل زمان پرس‌وجوی مساوی $O(\frac{m}{\sqrt{R}})$ می‌شود. زمان پیش‌پردازش $O(nR^2)$ می‌شود. چون برای هر کدام از R_iها $O(n)$ زمان برای تعیین مقدار دقیق جواب مسأله برای p_i صرف می‌شود زمان پیش‌پردازش در این مرحله $O(nR)$ است پس کل زمان پیش پردازش $O(nR)$ می‌شود. بنابراین جواب دقیق مسأله را می‌توان با حافظه‌ی $O(R)$ در زمان $O(\frac{m}{\sqrt{R}})$ پاسخ داد. توجه شود که در این قسمت منظور از زمان و جافظه‌ی گفته شده، مقدارهای مورد انتظار آنها هستند. بنابراین قضیه زیر را خواهیم داشت

قضیه ۵.۴ *با زمان پیش‌پردازش و حافظه‌ی مورد انتظار $O_\epsilon(nk)$ و $O_\epsilon(k)$ می‌توان مسأله‌ی شمارش را در زمان پرس و جوی مورد انتظار $O_\epsilon(\frac{m}{\sqrt{k}})$ به طور دقیق پاسخ داد* ($m \le k \le m^2$).

۷.۴ نتیجه‌گیری

در این فصل ابتدا الگوریتم تصادفی دیگری برای مسأله‌ی شمارش بیان کردیم. این الگوریتم مشابه روش فصل پیشین بود ولی تفاوت‌های اندکی نیز داشت. قابل ذکر است که زمان اجرای این الگوریتم وابسته به n است، در صورتیکه زمان اجرای الگوریتم فصل پیشین وابسته به m است. علاوه بر تحلیل ریاضی برای تحلیل تجربی، این الگوریتم را پیاده‌سازی و بر روی داده‌های تصادفی که تولید کرده بودیم نیز اجرا کردیم. نقطه‌ی قوت این الگوریتم پیاده‌سازی راحت و همین‌طور زمان اجرای مناسب آن است.

در رابطه با این الگوریتم و الگوریتم فصل پیشین باید به این نکته توجه کرد که به دلیل انتخاب تصادفی، مقدار مورد انتظار حافظه و زمان اجرا را محاسبه کردیم. قابل ذکر است که می‌توان به راحتی به‌طور قطعی نیز به این حافظه و زمان نیز رسید. برای این منظور کافی است به‌طور تصادفی انتخاب‌ها انجام شود ولی تعداد انتخاب‌ها همان مقدار مورد نظر ما باشد. در این حالت نیز جواب به‌دست آمده به‌صورت احتمالاتی خواهد بود.

در ادامه‌ی این فصل الگوریتمی برای محاسبه‌ی مقدار دقیق جواب مسأله‌ی شمارش بیان کردیم. زمان پرس‌وجوی این روش مشابه الگوریتم [۳۵] است. این الگوریتم بر خلاف روش‌های قبلی جواب دقیق مسأله را در همان حافظه و زمان پرس‌وجوی پاسخ می‌دهد. هر چند که زمان پیش‌پردازش این الگوریتم بیشتر از [۳۵] است. کاربردی بودن این الگوریتم را می‌توان به این صورت توجیه کرد که پیش‌پردازش فقط یک مرتبه انجام می‌شود و تنها چیزی که ذخیره می‌شود حافظه‌ی مورد نیاز است. در روش ارائه شده مسأله‌ی جستجوی عمومی بازه را در این حالت خاصی که برای مسأله‌ی ما پیش‌آمده

بود حل کردیم. از این ایده می‌توان برای حل حالت‌های مشابه و یا کلی‌تر نیز استفاده کرد. توجه داشته باشید که زمان پیش‌پردازش این روش با استفاده از ساده‌ترین الگوریتم‌هایی که وجود دارد بدست آمد. بدیهی است که می‌توان زمان پیش‌پردازش این روش را بهبود داد.

یکی از مسایلی که در آینده می‌توان بررسی کرد بهبود زمان پیش‌پردازش تا جایی است که مشابه روش [۳۵] باشد. به عبارتی با همان حافظه و زمان پیش‌پردازش مقدار دقیق جواب را بدست آوریم. به این نقطه باید توجه کرد که با در نظر گرفتن کران پایین مسأله‌ی هاپکرافت[۱] که در [۲۹] بیان شده است در [۳۵] اثبات شده‌است که این زمان به‌دست آمده توسط آنها نزدیک به زمان بهینه است. بنابراین الگوریتمی که بتواند مقدار دقیق مسأله را پاسخ دهد دارای پیچیدگی زمانی مشابه با زمانی که در [۳۵] ارائه شده است، خواهد بود.

[۱] Hopcroft's problem

فصل ۵

مسأله‌ی شمارش و نتایج تجربی در ۳ بعد

در این فصل مسایل مطرح شده را در فضای ۳ بعدی تعریف می‌کنیم. برخی قضیه‌های مطرح شده در فصل‌های قبل را به ۳ بعد تعمیم می‌دهیم. همچنین الگوریتم‌هایی که براساس فصل‌های قبلی هستند را ارائه می‌کنیم. برای بررسی کارایی این الگوریتم‌ها آنها را پیاده‌سازی و بر روی داده‌های واقعی اجرا و نتایج بدست آمده را نیز بیان می‌کنیم.

۱.۵ بیان مسأله‌ها در فضای ۳ بعدی

مسأله‌ی دیدپذیری و مسأله‌ی شمارش در فضای ۳ بعدی را به‌صورت زیر تعریف می‌کنیم:

سؤال ۱.۵ *مجموعه‌ی S از n مثلث در فضای ۳ بعدی داده شده است. می‌خواهیم به‌ازای نقطه‌ی پرس‌وجوی p و مثلث $\delta \in S$ تعیین کنیم که آیا p و $\delta \in S$ دیدپذیر هستند یا نه.*

سؤال ۲.۵ *مجموعه‌ی S از n مثلث در فضای ۳ بعدی داده شده است. می‌خواهیم به‌ازای نقطه‌ی پرس‌وجوی p تعداد مثلث‌های دیدپذیر را به‌دست آوریم.*

مشابه تعریف‌های پیشین مثلث Δ و نقطه p را دیدپذیر می‌گوییم اگر و فقط اگر نقطه‌ی q از Δ وجود داشته باشد که توسط p دیدپذیر باشد. همچنین مثلث دیدپذیر Δ را $edge - visible$ می‌گوییم اگر و فقط اگر نقطه‌ای روی یال‌ها یا راس‌های Δ وجود داشته باشد که توسط p دیدپذیر باشد. در غیر این‌صورت Δ را $mid - visible$ می‌گوییم.

در [۱۱] قضیه‌ای اثبات شده است که نشان می‌دهد تعداد پاره‌خط‌های دیدپذیر حداکثر ۲ برابر تعداد نقاط انتهایی دیدپذیر است. در این‌جا تعمیم این قضیه را در فضای ۳ بعدی به این‌صورت بیان

می‌کنیم.

قضیه ۳.۵ *تعداد مثلث‌های دیدپذیر حداکثر ۲ برابر تعداد مثلث‌های $edge - visible$ است.*

نتیجه ۴.۵ *با توجه به قضیه‌ی ۳.۵ تعداد مثلث‌های $edge - visible$ جوابی با ضریب تقریب ۲ برای مسأله‌ی شمارش بدست می‌دهد.*

مسأله‌ی مشابه دیگری که در عمل بیشتر مطرح می‌شود به این صورت است:

سؤال ۵.۵ *مجموعه‌ی T از n مثلث که تشکیل یک زمینه[۱] می‌دهند داده شده است. منظور از زمینه مجموعه‌ای از مثلث‌ها (۲ بعدی) در فضای ۳ بعدی است که در یال‌هایشان اشتراک دارند و هر خط عمود بر صفحه ی $x - y$ حداکثر در یک نقطه این مثلث‌ها را قطع می‌کند. می‌خواهیم به‌ازای نقطه‌ی پرس‌وجوی p تعیین کنیم که آیا مثلث داده شده‌ای توسط p دیده می‌شود و همینطور تعداد مثلث‌های دیدپذیر را به‌دست آوریم.*

تعریف ۶.۵ *مجموعه‌ی تمام نقاطی که توسط p دیده می‌شوند را منطقه‌ی دیدپذیر p می‌نامیم و با $VR(p)$ نشان می‌دهیم.*

محاسبه‌ی منطقه‌ی دیدپذیر یک نقطه می‌تواند نیاز به زمان $\Omega(n^2)$ داشته باشد[۲۰، ۲۲]، که n تعداد مثلث‌های موجود در T است. بنابراین ارائه‌ی الگوریتم‌های تقریبی که مقدار تقریبی از $VR(p)$ را در زمان کمتری محاسبه می‌کنند مورد توجه است. در [۴۴] الگوریتم رادار مانندی برای محاسبه‌ی $VR(p)$ ارائه شده است. آنها ۴ الگوریتم با نام‌های $ECH - Fixed$، ECH، $Fixed - radar -$ و $like$ و $Radar - like$ ارائه داده‌اند. نتایج تجربی آنها نشان می‌دهد که الگوریتم $Radar - like$ سریعترین الگوریتم است. در ادامه الگوریتم‌های جدیدی برای این مسأله ارائه می‌دهیم که پایه‌ی آنها براساس نتایج فصل‌های پیشین است. همچنین با پیاده‌سازی این الگوریتم‌ها، نتایج بدست آمده را با الگوریتم‌های ارائه شده در [۴۴] مقایسه می‌کنیم.

۲.۵ مسأله‌ی دیدپذیری

ابتدا لم زیر را که بر اساس تعریف‌های فصل‌های پیشین هست، بیان می‌کنیم.

[۱] Terrain

لم ۷.۵ *اگر مثلث‌های داده شده تشکیل یک زمینه بدهند در این‌صورت تمامی مثلث‌های دیدپذیر* $edge - visible$ *خواهند بود.*

بنابراین بر اساس لم ۷.۵ برای محاسبه‌ی تعداد مثلث‌های دیدپذیر کافی است فقط یال‌های مثلث‌ها را در نظربگیریم و نیازی به بررسی نقاط روی سطح مثلث‌ها نداریم.

۱.۲.۵ الگوریتم ارائه شده برای مسأله‌ی دیدپذیری

برای هر مثلث Δ_1، از یکی از راس‌های آن، مثلا a_1 و یکی از یال‌هایی که a_1 برروی آن قرار دارد شروع می‌کنیم. اگر a_1 و p دیدپذیر باشند، در این‌صورت Δ_1 دیدپذیر خواهد بود و الگوریتم پایان می‌یابد. در غیراین‌صورت مثلث Δ_2ای که تصویرش بر روی a_1 می‌افتد وجود دارد که a_1 را می‌پوشاند. Δ_2 قسمتی از یالی که a_1 بر روی آن قرار دارد را می‌پوشاند. برای قسمت باقیمانده اولین نقطه a_2 بر روی آن یال را در نظر می‌گیریم که توسط Δ_2 پوشیده نمی‌شود. اگرa_2 و p دیدپذیر باشند در این‌صورت Δ_1 و p دیدپذیر خواهند بود. در غیر این‌صورت یک شعاع از a_2 به سمت p ساطع می‌کنیم و اولین مثلثی که قطع می‌کند را در نظر می‌گیریم. در ادامه الگوریتم را به همین ترتیب که بر روی Δ_2 اجرا کردیم ادامه می‌دهیم. اگر به انتهای یال رسیدیم در این‌صورت آن یال دیدپذیر نیست. به همین ترتیب الگوریتم را بر روی هر ۳ یال مثلث اجرا می‌کنیم.

زمان اجرای این الگوریتم برای هر نقطه‌ی پرس‌وجوی p و مثلث Δ_1 به تعداد شعاع‌های ساطع شده که مساوی تعداد مثلث‌های پوشاننده‌ی یال‌های Δ_1 است بستگی دارد. این عدد را با $t_p(\Delta_1)$ نشان می‌دهیم. بنابراین زمان اجرای الگوریتم $O(t_p(\Delta)f(n))$ است که $f(n)$ زمان ساطع شدن هر شعاع است.

۲.۲.۵ نتایج تجربی برای مسأله‌ی قابلت دید

در داده‌هایی که ما بررسی می‌کنیم به ازای هر نقطه مختصات (i,j) بر روی یک شبکه، ارتفاع آن نقطه مشخص شده‌است. همچنین شبکه‌ی داده شده مثلث‌بندی شده و نمایانگر یک زمینه است. برای هر ۴ نقطه‌ی $x_1 = (i,j,k_1)$ ، $x_2 = (i+1,j,k_2)$ ، $x_3 = (i,j+1,k_3)$ و $x_4 = (i+1,j+1,k_4)$ دو مثلث در نظر گرفته می‌شود. $\Delta_1 = (x_1,x_2,x_4)$ و $\Delta_2 = (x_1,x_3,x_4)$. در الگوریتمی که اجرا می‌کنیم بر روی مثلث‌ها پیش‌پردازشی انجام نمی‌شود، بنابراین $f(n) = O(\sqrt{n})$. برای هر مجموعه داده تعدادی نقطه به‌طور تصادفی به عنوان نقطه‌ی پرس‌وجو انتخاب کرده‌ایم و الگوریتم **؟؟** را برای

هر کدام از نقطه‌ها و تمام مثلث‌های زمینه اجرا کرده‌ایم. برای هر مثلث $\Delta_1 \in T$ ، مقدار $t_p(\Delta_1)$ را محاسبه می‌کنیم. نتایج تجربی نشان می‌دهد که:

$$E(t_p(\Delta_1)) = \sum_{\Delta_i \in T}^{n} t_p(\Delta_i)/n = O(1).$$

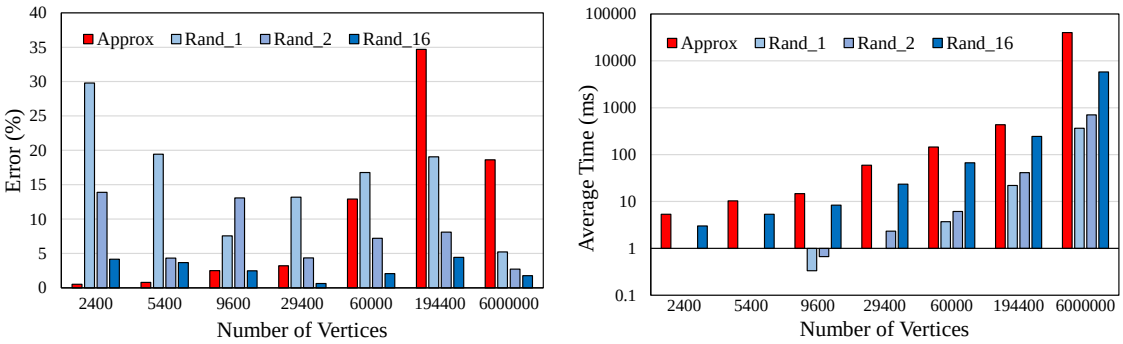
این نتیجه می‌دهد که میانگین تعداد مثلث‌هایی که یک مثلث تصادفی را می‌پوشانند به ازای یک نقطه‌ی تصادفی داده شده $O(1)$ است. یعنی میانگین زمان اجرا برای مسأله‌ی دیدپذیری بین یک نقطه تصادفی و یک مثلث، $O(f(n))$ است. جدول ۳.۵ میانگین t_p را برای هر کدام از مجموعه داده‌ها نشان می‌دهد. همچنین این نتایج نشان می‌دهد که میانگین $t_p(\Delta)$ مستقل از اندازه‌ی مجموعه داده است.

تعداد مثلث‌ها	میانگین مثلث‌های پوشاننده
۲,۴۰۰	۱٫۷۰
۲,۴۰۰	۱٫۱۸
۲,۴۰۰	۱٫۰۶
۵,۴۰۰	۱٫۵۸
۵,۴۰۰	۱٫۰۹
۵,۴۰۰	۱٫۰۳
۹,۶۰۰	۱٫۵۳
۹,۶۰۰	۱٫۱۰
۹,۶۰۰	۱٫۰۴
۲۹,۴۰۰	۱٫۴۸
۲۹,۴۰۰	۱٫۰۲
۲۹,۴۰۰	۱٫۰۸
۶۰,۰۰۰	۱٫۵۱
۶۰,۰۰۰	۱٫۳۷
۶۰,۰۰۰	۱٫۰۸

جدول ۱.۵ : میانگین تعداد مثلث‌هایی که یک مثلث را می‌پوشانند برای یک نقطه‌ی تصادفی برای هرکدام از مجموعه داده‌ها.

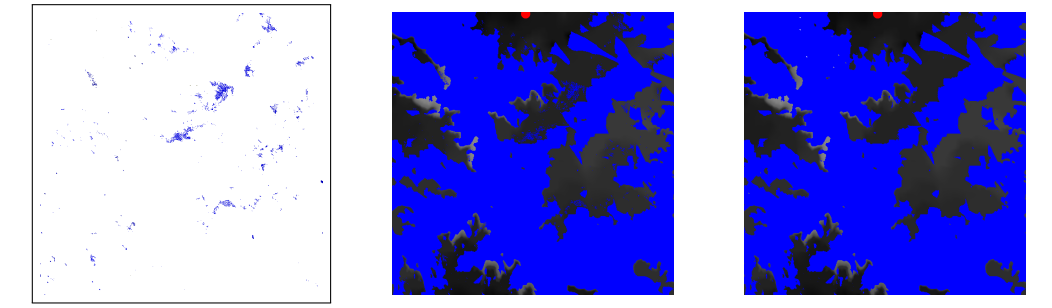
۳.۵ مسأله‌ی شمارش

همانطور که گفتیم برای پیدا کردن جواب دقیق مسأله‌ی شمارش مثلث‌های دیدپذیر زمینه کافی است مثلث‌های $edge - visible$ را بشماریم.



(آ) میانگین زمان اجرا (ب) میانگین خطا

شکل ۱.۵: میانگین زمان اجراو میانگین خطا برای الگوریتم‌های تصادفی و تقریبی.



(آ) منطقه‌ی قایل‌دبد تقریب زده شده (ب) منطقه‌ی دیدپذیر دقیق (ج) تقاضل ناحیه‌ی دیدپذیر دقیق و تقریبی

شکل ۲.۵: منطقه‌ی دیدپذیر نقطه‌ی پرس‌وجوی $p(۳/۳, ۲۴۲/۵, ۲۰۸۹/۵)$ (الف) تقریبی و (ب) دقیق. نقطه‌ی پرس‌وجو با رنگ قرمز نشان داده شده‌است.

۱.۳.۵ الگوریتم تقریبی برای مسأله‌ی شمارش

در این بخش الگوریتم تقریبی برای شمارش تعداد مثلث‌هایی که حداقل یکی از راس‌هایشان دیدپذیر است ارائه می‌دهیم. در این الگوریتم ابتدا شعاعی دلخواه از نقطه‌ی داده شده p به سمت زمینه ساطع می‌کنیم. واضح است که مثلث Δ_1 که توسط این شعاع قطع می‌شود توسط p دیدپذیر است. در مرحله‌ی بعدی شعاع‌هایی به سمت راس‌های Δ_1 ساطع می‌کنیم. در این صورت اولین مثلث‌هایی که توسط این شعاع‌ها قطع می‌شوند توسط p دیدپذیر هستند. به همین ترتیب الگوریتم را بر روی آن مثلث‌ها اجرا می‌کنیم و به طور تقریبی تعداد مثلث‌های دیدپذیر را به‌دست می‌آوریم (الگوریتم ۱).

۲.۳.۵ پیچیدگی زمانی

در الگوریتم ۱ هر مثلثی فقط یکبار وارد صف می‌شود. برای هر مثلث ۳ شعاع به سمت راس‌هایش ساطع می‌کنیم. در هر مرحله از ساطع کردن شعاع بررسی می‌کنیم که آیا مثلثی بین p و یک راس وجود

Algorithm 1 Approximation algorithm for computing the visible triangles

Input:
T: A 2D array of triangles.
e: A query point p.
Output:
The number of visible segments from p.
Method:

```
1: Create empty Queue, bfsQueue.
2: Find triangle Δ1 intersected by the ray emanating from p to the surface of terrain.
3: Add Δ1 to bfsQueue.
4: while bfsQueue is not empty do
5:    Δt = bfsQueue.pop()
6:    if At least one vertex of triangle Δt is visible from p then
7:       Add three neighbors of Δ1 to bfsQueue.
8:    else
9:       Find concealer triangles that cover vertices of triangle Δt.
10:      Shoot rays from p to the vertices of concealer triangles
11:      Find hit points of shooting ray and the surface of the terrain.
12:      Add triangles corresponding to the hit points to bfsQueue.
13:   end if
14: end while
```

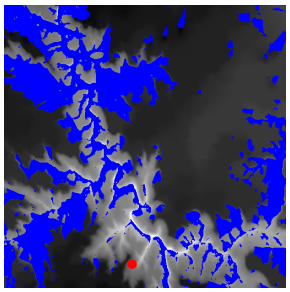
دارد یا نه. برای هر شعاع می‌خواهیم اولین مثلثی که توسط شعاعی که از p ساطع شده قطع می‌شود را بیابیم. به ازای هر شعاعی که ساطع می‌شود $O(\sqrt{n})$ مثلث را بررسی می‌کنیم. بنابراین زمان اجرای الگوریتم $O(\sqrt{n}m_p)$ می‌شود که m_p تعداد مثلث‌های دیدپذیر است.

۳.۳.۵ الگوریتم تقریبی تصادفی برای مسأله‌ی شمارش

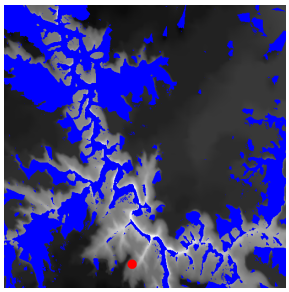
حال الگوریتم تصادفی برای تقریب جواب مسأله‌ی شمارش ارائه می‌دهیم. این الگوریتم بر اساس نمونه‌گیری تصادفی و مشابه روش‌های قبلی است. زمان اجرای این الگوریتم $O(\frac{1}{\delta^2}\sqrt{n}\ f(n)\ \log n)$ است. که در اینجا δ ضریب تقریب الگوریتم است. تحلیل زمان اجرا و ضریب تقریب و میزان خطای این الگوریتم همانند الگوریتم‌های فصل‌های قبل است که از لم ۹.۳ استفاده کرده بودیم.

۴.۳.۵ الگوریتم دقیق مسأله‌ی شمارش

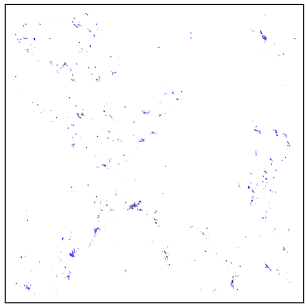
با استفاده از الگوریتمی که برای تعیین دیدپذیری مثلث‌ها بیان کردیم می‌توانیم مسأله‌ی دیدپذیری را برای هر کدام از مثلث‌ها حل کنیم و در نتیجه مقدار دقیق پاره‌خط‌های دیدپذیر را بدست آوریم. زمان اجرای این الگوریتم مهم نیست و فقط برای بررسی کارایی الگوریتم تقریبی استفاده می‌شود.



(آ) منطقه‌ی قایل‌دبد تقریب زده شده



(ب) منطقه‌ی دیدپذیر دقیق



(ج) تقاضل ناحیه‌ی دیدپذیر دقیق و تقریبی

شکل ۳.۵: منطقه‌ی دیدپذیر نقطه‌ی پرس‌وجوی $p(۴۶۸/۸, ۲۳۲/۵, ۳۲۲۸/۶)$ (الف) تقریبی و (ب) دقیق. نقطه‌ی پرس‌وجو با رنگ قرمز نشان داده شده‌است.

۴.۵ نتایج تجربی مسأله‌ی شمارش

الگوریتم تقریبی ارائه شده را بر روی ۳ مجموعه داده اجرا کرده‌ایم.

شکل ۲.۵ و۳.۵ مقدار دقیق و تقریبی مناطق دیدپذیر ۲ نقطه‌ی پرس و جو را نشان می‌دهند. ما مقدار تقریبی ناحیه دیدپذیر هر نقطه را با استفاده از الگوریتم ۱ محاسبه می‌کنیم. سپس مقدار دقیق را محاسبه می‌کنیم. همان‌طور که دیده می‌شود مقدار تقریبی به مقدار دقیق نزدیک است.

برای بررسی کارایی الگوریتم، تابع خطایی به این صورت تعریف می‌کنیم. اگر m'_p تعداد مثلث‌های دیدپذیر تقریب‌زده شده‌باشد در این‌صورت مقدار خطا $\frac{(m_p - m'_p)}{m_p}$ است. برای هر مجموعه داده تعداد ۱۰۰۰ نقطه‌ی تصادفی با ارتفاع‌های مختلف انتخاب کرده‌ایم. سپس مقدار خطا را برای هر نقطه محاسبه کرده‌ایم. واضح است که با افزایش ارتفاع، تعداد مثلث‌های دیدپذیر و بنابراین زمان اجرای الگوریتم افزایش می‌یابد. جدول ۲.۵ متوسط زمان اجرا و همینطور متوسط خطا را برای هر مجموعه داده نشان می‌دهد.

ما نتایج خود را با نتایج [۴۴] مقایسه می‌کنیم. آنها ۴ الگوریتم ارائه کرده‌اند. ما نیز مجموعه داده‌ای مشابه مجموعه داده‌های آنها به کار برده‌ایم. آنها ۱۰ مجموعه داده را که هر کدام منطقه‌ی جغرافیایی خاصی را مشخص می‌کنند، استفاده کرده‌اند. هر زمینه‌ای به صورت اجتماع $۵,۰۰۰ - ۱۰,۰۰۰$ مثلث است. برای هر زمینه‌ای نقاط مختلفی را به صورت تصادفی انتخاب کرده‌اند . برای هر نقطه‌ای هر کدام از آن ۴ الگوریتم را برای محاسبه‌ی منطقه‌ی دیدپذیر آن نقطه استفاده کرده‌اند. همچنین میزان خطا را نیز بر اساس تابع خطایی که تعریف کرده‌اند نیز اندازه گرفته‌اند.

تابع خطایی که آنها استفاده کرده‌اند به این صورت تعریف شده‌است. اگر R'_p مقدار تقریبی R_p که R_p منطقه‌ی دیدپذیر توسط p است باشد در این‌صورت خطای نسبت داده شده، یای انحصاری R_p

Algorithm 2 Randomized Algorithm for VCP

Input:
Triangles of terrain T.
p: a query point.
t: number of execution iterations of randomized algorithm.
Output:
counter: the number of triangles of terrain T visible from p
Method:

```
set p to 1/√n where n is number of triangles in terrain T.
set counter to 0.
loop
   t iterations
   for each triangle Δ of terrain T do
      select triangle Δ with probability of p.
      use VTP algorithm to check whether triangle Δ is visible from p or not.
      if Δ is visible from p then
         counter++
      end if
   end for
end loop
return counter/(t×p)
```

و R'_p تقسیم بر کل ناحیه‌ی زمینه است.

مجموعه داده‌هایی که آنها استفاده کرده‌اند متشکل از ۱۰,۰۰۰ – ۵,۰۰۰ مثلث بوده‌است. ما نیز از مجموعه داده‌هایی متشکل از ۱۰,۰۰۰ مثلث استفاده کرده‌ایم.

همانطور که در جدول ۳.۵دیده می‌شود میانگین بهترین زمان اجرای الگوریتم آنها ۲۷۴ میلی ثانیه با میزان خطای ۰٫۵ (با تابع خطای آنها) ومیانگین بهترین زمان اجرای ما ۷۰ میلی ثانیه با خطای ۰٫۵۵ (با استفاده از تابع خطای خودمان) است. اگر ما از تابع خطای آنها استفاده کنیم خطا ۰٫۳۵ خواهد بود.

مقایسه‌ی سیستم‌هایی که الگوریتم‌ها بر روی آنها اجرا شده‌اند در جدول ۴.۵ آمده است.

[44]	Pentium 4	2.4GHz	Linux 8.1	Java 1.4
Our platform	Core i5	2.4GHz	Win10	Java 1.8

جدول ۴.۵: مقایسه‌ی سیستم‌ها

توجه کنید که زمان اجرای هر دو الگوریتم ارائه شده بهتر از [۴۴] است. مزیت الگوریتم اول در این است که می‌توانیم منطقه‌ی دیدپذیر p را با خطای کمتری نسبت به [۴۴] محاسبه کنیم. هر

تعداد مثلثها	زمان اجرا (ms)	خطا
۲,۴۰۰	۱۷	۱/۳۹
۲,۴۰۰	۱۴	۰/۴۵
۲,۴۰۰	۱۳	۰/۴۱
۵,۴۰۰	۴۴	۲/۰۳
۵,۴۰۰	۳۶	۰/۸۹
۵,۴۰۰	۲۸	۰/۴۵
۹,۶۰۰	۹۰	۲/۲۸
۹,۶۰۰	۸۰	۱/۱۷
۹,۶۰۰	۶۶	۰/۹۰
۲۹,۴۰۰	۳۵۵	۳/۹۲
۲۹,۴۰۰	۳۹۶	۱/۷۸
۲۹,۴۰۰	۳۶۵	۲/۴۷
۶۰,۰۰۰	۸۴۰	۵/۲۸
۶۰,۰۰۰	۸۷۰	۳/۱۷
۶۰,۰۰۰	۹۵۴	۳/۷۰

جدول ۲.۵: میانگین زمان اجرای الگوریتم به ازای هر کدام از مجموعه داده‌ها.

Error	1.00	0.75	0.5
Fixed ECH	597	1,0045	1,648
ECH	579	1,012	1,591
Fixed radar-like	112	192	301
Radar-like	101	168	274

جدول ۳.۵: نتایج [۴۴]. آنها تابع خطایی برای تقریب منطقه‌ی دیدپذیر یک نقطه‌ی داده شده ارائه داده‌اند. زمان اجرای الگوریتم آنها برای هر مقدار خطا داده شده است. بهترین الگوریتم، الگوریتم $Radar-like$ است.

چند که زمان اجرای الگوریتم دوم خیلی بهتر از الگوریتم اول است ولی ما در این الگوریتم منطقه‌ی دیدپذیر را محاسبه نمی‌کنیم و فقط نسبت ناحیه‌ای که توسط p دیده می‌شود را محاسبه می‌کنیم.

۵.۵ نتیجه‌گیری

در این فصل مسأله‌ی دید و شمارش را در فضای ۳ بعدی تعریف کردیم و الگوریتم‌هایی بر اساس فصل‌های قبل ارائه دادیم. همچنین این الگوریتم‌ها را پیاده‌سازی و بر روی داده‌های واقعی اجرا کردیم. نکته‌ی قابل توجه زمان اجرای کم و مقدار خطای کم این الگوریتم‌ها در مقایسه با الگوریتم‌های پیشین بود. شایان ذکر است که به دلیل عدم دسترسی به داده‌هایی که در مقاله‌های مرجع ما استفاده شده‌بود ما نتوانستیم الگوریتم‌هایمان را بر روی آن داده‌ها اجرا کنیم و بنابراین الگوریتم‌ها را بر روی داده‌هایی مشابه از نظر اندازه اجرا کردیم. پیاده‌سازی دیگر الگوریتم‌های فصل‌های پیشین و همینطور بررسی

بیشتر مسأله‌ها از جمله تعمیم قضایای فصل‌های پیشین به ۳ بعد از مطالبی است که می‌توان در آینده در نظر گرفت.

فصل ۶

مدل احتمالاتی

در این فصل یک مدل تصادفی جدید از مسأله‌ی دید و مسأله‌ی شمارش تعریف می‌شود. فرض کنید $\mathcal{S}$ مجموعه‌ای از n پاره‌خط نادقیق[۱] باشد. به بیان دقیق‌تر برای هر پاره‌خط $s_i \in \mathcal{S}$ یک تابع توزیع احتمالی گسسته در نظر می‌گیریم به‌طوریکه یک مجموعه $\mathcal{D}_i = \{s_{i,1}, \cdots, s_{i,m_i}\} \cup \{s_{i,\cdot} = \perp\}$ از مکان‌های ممکن با احتمال‌های نسبت داده شده‌ی $p_{i,j}$ داریم و $\mathbf{Pr}(s_i = s_{i,j}) = p_{i,j}$ و $\sum_j p_{i,j} = 1$. علامت $\perp$ به این معنی است که پاره‌خط s_i در $\mathcal{S}$ وجود ندارد. در این جا مجموعه‌ی $\mathcal{S}$ را می‌توان به صورت یک متغیر تصادفی دید چون شامل پاره‌خط‌های احتمالی است. متغیر تصادفی مقادیر خود را از یک فضای نمونه با اندازه‌ی $\Pi_i(m_i + 1)$ با احتمال مساوی $\Pi_{s \in \mathcal{S}} \mathbf{Pr}(s) \Pi_{s \notin \mathcal{S}}(1 - \mathbf{Pr}(s))$ می‌گیرد. فرض کنید که $z = \max\{1 + m_i\}$. در حالت خاص که $z = 2$ نادقیقی در رابطه با وجود و عدم وجود پاره‌خط‌ها است، نه در رابطه با مکان آنها.

براساس این مطالب حالت احتمالاتی مسأله‌های دیدپذیری و شمارش را تعریف می‌کنیم.

سؤال ۱.۶ *مسأله‌ی دیدپذیری احتمالاتی: به ازای پاره‌خط داده شده‌ی $s \in \mathcal{S}$ احتمال دیده شدنش توسط نقطه‌ی پرس‌وجوی q را محاسبه کنید.*

سؤال ۲.۶ *مسأله‌ی شمارش احتمالاتی: امید ریاضی تعداد پاره‌خط‌هایی را که توسط نقطه‌ی پرس‌وجوی q دیدپذیر هستند، محاسبه کنید.*

در ادامه‌ی این فصل مسأله‌های عنوان شده را بررسی می‌کنیم.

[۱] Uncertain

۱.۶ مسأله‌ی دید احتمالاتی

همان‌طور که می‌دانیم مسأله‌ی شمارش تعداد تطابق‌ها در یک گراف ۲ بخشی از مسایل معروف $\#P-complete$ است [۵۲]. در ابتدا مسأله‌ی شمارش تعداد تطابق‌های کامل یک گراف ۲ بخشی را به این دیدپذیری احتمالاتی کاهش می‌دهیم تا نشان دهیم مسأله‌ی دید احتمالاتی $\#P-complete$ است.

فرض کنید می‌خواهیم تعداد تطابق‌های کامل گراف ۲ بخشی $G=(U,V,E)$ را به‌دست آوریم. که $U=\{u_1,\cdots,u_n\}$ و $V=\{v_1,\cdots,v_n\}$ مجموعه‌ی راس‌ها و E مجموعه‌ی یال‌های G هستند. برای گراف دو بخشی داده شده یک نمونه از مسأله دیدپذیری احتمالاتی را به شکل زیر می‌سازیم به‌طوریکه هر تطابق کامل به‌طور یکتا متناظر با یک نمونه از فضای نمونه‌ی پاره‌خط‌های نامعین است که در آن s توسط q دیده نمی‌شود. تعداد n بازه $[i,i+1]$ بر روی محور x در نظر بگیرید. که i از ۰ تا $n-1$ تغییر می‌کند. فرض کنید که بازه‌ی $[i,i+1]$ متناظر با راس v_i است و با $I(v_i)$ نشان می‌دهیم. برای هر راس $u_i\in U$، یک پاره‌خط نادقیق $\mathcal{D}_i=\{I(v_j)|\{u_i,v_j\}\in E\}$ با توزیع یکنواخت تعریف می‌کنیم. توجه کنید که در این نمونه هر پاره‌خط نادقیق همواره وجود دارد. یک پاره‌خط نادقیق s با احتمال یک که نقاط انتهایی‌اش $(0,-1)$ و $(n,-1)$ است نیز در نظر می‌گیریم. نقطه‌ی q را بالای محور در نظر بگیرید. پاره‌خط s توسط q دیده نمی‌شود اگر و فقط اگر $[0,n]$ کاملا توسط پاره‌خط‌های نادقیق پوشیده شود. تعداد n پاره‌خط نادقیق وجود دارد و هرکدام دقیقا یک واحد از $[0,n]$ را می‌پوشاند. بنابراین هر پاره‌خط نادقیق دقیقا باید یک واحد از بازه‌ها را بپوشاند.

این شهود اصلی در تناظر یک به یک بین تطابق کامل و زیرمجموعه‌ی فضای نمونه که در آن s توسط q دیدپذیر نیست، است. بنابراین قضیه‌ی زیر را خواهیم داشت.

قضیه ۳.۶ *مسأله‌ی دیدپذیری احتمالاتی $\#P-complete$ است.*

در این بخش حالت $z=2$ را در نظر می‌گیریم. به عبارتی هر پاره‌خطی یا وجود ندارد یا فقط در یک مکان وجود دارد. بنابراین می‌توان فرض کرد که n پاره‌خط نادقیق $s_1,\cdots,s_n$ داریم و $\mathbf{Pr}(s_i\in\mathcal{S})=p_i$ که نتیجه می‌دهد $\mathbf{Pr}(s_i\notin\mathcal{S})=1-p_i$.

در ادامه نشان می‌دهیم که برای یک پاره‌خط داده شده s و نقطه‌ی q چگونه مقدار $pr(q\ sees\ s)$ را محاسبه کنیم. واضح است که اگر $s\notin\mathcal{S}$ باشد، q نمی‌تواند s را ببیند.

پس $pr(q\ sees\ s)=\mathbf{Pr}(q\ sees\ s|s\in\mathcal{S})\mathbf{Pr}(s\in\mathcal{S})$. بنابراین $\mathbf{Pr}(q\ sees\ s|s\in\mathcal{S})$ را باید محاسبه کنیم.

فرض کنید Δ مثلثی باشد با راس q و یال s. هر پاره‌خط نادقیق که Δ را قطع نمی‌کند تاثیری در محاسبه‌ی احتمال ندارد. بنابراین فقط پاره‌خط‌هایی را در نظر می‌گیریم که Δ را قطع می‌کنند. تصویر این پاره خط‌ها را بر روی s تحت q در نظر بگیرید. در این صورت مسأله‌ای که حل می‌کنیم به این صورت خواهد بود.

- فرض کنید تعداد n بازه‌ی نادقیق $I = \{I_۱, \cdots, I_n\}$ بر روی محور اعداد حقیقی داده شده است. هر بازه‌ی I_i با احتمال p_i وجود دارد. حال احتمال پوشیده شدن بازه‌ی $[a, b]$ را که با $pr([a, b]\ is\ covered\ )$ نشان می‌دهیم، به‌دست آورید.

به نظر می‌رسد که محاسبه‌ی احتمال خواسته شده به زمان $\Theta(۲^n)$ نیاز دارد چون اندازه‌ی فضای نمونه می‌تواند در بدترین حالت از $\Theta(۲^n)$ باشد. اما در ادامه نشان می‌دهیم که در زمان $O(n \log n)$ می‌توان این محاسبات را انجام داد. برای سادگی می‌توان فرض کرد که بازه‌ها بر اساس نقاط انتهایی سمت راستشان مرتب شده‌اند. فرض کنید $r(I_i)$ ($l(I_i)$) نقطه‌ی انتهایی سمت راست (چپ) I_i باشد. فرمول بازگشتی ارائه می‌دهیم. برای هر نقطه‌ی $a' \in [a, b]$ فرض کنید $sol(a')$ احتمال پوشیده شدن $[a', b]$ باشد. بنابراین $sol(a)$ احتمال پوشیده شدن $[a, b]$ خواهد بود. $S(a') = \{I'_۱, ..., I'_l\}$ را مجموعه‌ی تمام بازه‌هایی در نظر بگیرید که a' را می‌پوشانند و بر اساس نقطه‌ی انتهایی راستشان مرتب شده‌اند. بر اساس این تعریف‌ها خواهیم داشت:

لم ۴.۶ ۱ $sol(b) =$ *تعریف می‌کنیم. در این‌صورت خواهیم داشت:*

$$sol(a') = \sum_{j=۱}^{l} p'_j(\prod_{i=۱}^{j-۱}(۱ - p'_i))sol(r(I'_j)).$$

اثبات. فرض کنید $a' \in [a, b]$، بنابراین اگر $[a', b]$ پوشیده شود در این‌صورت حداقل یکی از پاره‌خط‌های موجود در $S(a')$ باید انتخاب شوند. تعداد l پاره‌خط وجود دارند که a' را می‌پوشانند. از آن‌جایی که پاره‌خط‌های $S(a')$ بر اساس نقطه‌ی انتهایی سمت چپشان مرتب شده‌اند آنگاه احتمال این‌که I'_j اولین پاره‌خطی باشد که a' را می‌پوشاند مساوی $p'_j \prod_{i'=۱}^{j-۱}(۱-p'_i)$ خواهد بود. به‌طور بازگشتی $[a', b]$ با احتمال $sol(r(I'_j))$. بنابراین خواهیم داشت: $sol(a') = \sum_{j=۱}^{l} p'_j(\prod_{i'=۱}^{j-۱}(۱ - p'_i))sol(r(I'_j))$.

■

هرکدام از نقاط انتهایی‌بازه‌ها می‌توانند توسط $O(n)$ بازه‌ی دیگر پوشیده شوند. در فرمول بازگشتی هر نقطه‌ی انتهایی حداکثر یکبار صدا زده می‌شود. برای هر $sol(r(I'_j))$ باید مقدار $\prod_{i'=۱}^{j-۱}(۱ - p'_i)$ را محاسبه کنیم. از آن‌جایی پاره‌خط‌ها بر اساس نقاط انتهایی‌شان مرتب شده‌اند، برای هر $sol(r(I'_j))$

مقدار $\prod_{i'=1}^{j-2}(1-p'_i)$ را در $1-p'_j$ ضرب می‌کنیم. بنابراین می‌توانیم $sol(a)$ را در زمان $O(n^2)$ محاسبه کنیم. در ادامه الگوریتم سریعتری را بیان می‌کنیم. برای پر کردن آرایه‌ی sol، نقاط انتهایی را از راست به چپ جاروب می‌کنیم. نقاطی را که خط جاروب را قطع می‌کنند را در یک درخت جستجوی دودویی ذخیره می‌کنیم. همچنین در زمان پردازش مقدار اضافه‌ی دیگری را نیز در هر گره به‌صورتی که گفته می‌شود ذخیره می‌کنیم. در زمان پردازش یک نقطه‌ی انتهایی راست، $r(I_i)$ مقدار$sol(r(I_i))$ را محاسبه می‌کنیم که در این لحظه مساوی مجموع تمام گره‌های درخت است. این کار را می‌توان در زمان $O(\log n)$ انجام داد. سپس به‌طور ضمنی تمامی گره‌ها را در $(1-p_i)$ ضرب می‌کنیم. سپس $r(I_i)$ را با مقدار $p_i sol(r(I_i))$ به درخت اضافه می‌کنیم. این کار را نیز می‌توان در زمان $O(\log n)$ انجام داد. تعداد کل نقاط انتهایی $O(n)$ است و بنابراین زمان اجرای این الگوریتم $O(n\log n)$ خواهد بود. پس قضیه‌ی زیر را خواهیم داشت.

قضیه ۵.۶ *در حالت $z=2$، مسأله‌ی دیدپذیری احتمالاتی را می‌توان در زمان $O(n\log n)$ پاسخ داد.*

حال روشی ارائه می‌دهیم که با استفاده از پیش‌پردازش برروی پاره‌خط‌ها بتوانیم مسأله را در زمان $O(\log n)$ پاسخ دهیم. همانطور که قبلا هم گفته شد فرض کنید تمامی پاره‌خط‌ها داخل یک مربع B قراردارند. در ابتدا هر جفت از نقاط انتهایی را به هم وصل می‌کنیم و این خط را ادامه می‌دهیم تا B را قطع کند. این خط‌ها B را به $O(n^4)$ ناحیه افراز می‌کنند. برای هر پاره‌خط $s\in\mathcal{S}$، جواب مسأله دید احتمالاتی برای تمامی نقاط موجود در یک ناحیه یکسان است. چون ترتیب ترکیبیاتی پاره‌خط‌هایی که s را می‌پوشانند برای تمامی نقاط یک ناحیه یکسان است. بنابراین در زمان پیش‌پردازش به ازای هر ناحیه r_i یک نقطه‌ی دلخواه q_i از آن ناحیه انتخاب می‌کنیم و مقدار $\mathbf{Pr}(q_i\ sees\ s)$ رادر زمان $O(n\log n)$ محاسبه می‌کنیم. بنابراین به ازای پاره‌خط‌های داده شده $\mathcal{S}$ و پاره‌خط $s\in\mathcal{S}$، در زمان $O(n^5\log n)$ و با حافظه‌ی $O(n^4)$ پیش‌پردارش انجام می‌دهیم به طوریکه به ازای هر نقطه‌ی پرس‌وجوی q، می‌توانیم ناحیه‌ی r_i شامل q را در زمان $O(\log n)$ بدست آوریم و مقدار $\mathbf{Pr}(q_i\ sees\ s)=\mathbf{Pr}(q\ sees\ s)$ را محاسبه کنیم.

۲.۶ مسأله‌ی شمارش احتمالاتی

در این بخش مسأله‌ی شمارش احتمالاتی را بررسی می‌کنیم. ابتدا تعریفی اولیه بیان می‌کنیم. برای هر زیر مجموعه $T\subset\mathcal{S}$، فرض کنید $m_q(T)$ تعداد پاره‌خط‌هایی هستند از مجموعه‌ی T که توسط q دیده می‌شوند. بنابراین مجموعه‌ی پاره‌خط‌های دیدپذیر را می‌توان به این‌صورت نوشت.

$$E(m_q) = \sum_{T \subseteq \mathcal{S}} \mathbf{Pr}(T) m_q(T),$$

که $\mathbf{Pr}(T)$ احتمال این است که مجموعه‌ی انتخاب شده T باشد. با استفاده از خطی بودن امید ریاضی نیز می‌توان $E(m_q)$ را محاسبه کرد.

$$E(m_q) = \sum_{i=1}^{n} \mathbf{Pr}(q \ sees \ s_i).$$

برای حالت $z = 2$، می‌توانیم از تساوی بالا و الگوریتم بخش قبل استفاده کنیم و $E(m_q)$ را در زمان $O(n^2 \log n)$ محاسبه کنیم. همچنین همانند بخش قبل با پیش‌پردازش پاره‌خط‌ها در زمان $O(n^6 \log n)$ و حافظه‌ی $O(n^4)$ می‌توانیم مسأله را در زمان $O(\log n)$ پاسخ دهیم. توجه کنید که جواب دقیق مسأله را فقط برای حالت $z = 2$ به‌دست می‌آوریم. حال نشان می‌دهیم که چگونه جواب تقریبی مسأله را برای حالت کلی در زمان بهتر به‌دست آوریم.

۱.۲.۶ الگوریتم تقریبی برای مسأله‌ی شمارش احتمالاتی

دراین بخش الگوریتمی با ضریب تقریبی ۲ برای مسأله‌ی شمارش احتمالاتی بیان می‌کنیم. با توجه به [۱۱] اگر T مجموعه‌ی نمونه باشد و $m_q(T)$ و$ve_q(T)$ به ترتیب تعداد پاره‌خط‌های دیدپذیر و نقاط انتهایی دیدپذیر در $T \subset \mathcal{S}$ باشند خواهیم داشت:

$$m_q(T) \leq ve_q(T) \leq 2m_q(T).$$

پس می‌توانیم نتیجه بگیریم:

$$\sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) m_q(T) \leq \sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) ve_q(T) \leq \sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) 2 m_q(T).$$

یا به‌عبارت دیگر،

$$E(m_q) \leq E(ve_q) \leq 2E(m_q).$$

بنابراین مقدار $E(ve_q) = \sum_{i=1}^{n} \mathbf{Pr}(r(s_i) \ sees \ q) + \mathbf{Pr}(l(s_i) \ sees \ q)$ را محاسبه می‌کنیم.

$$\mathbf{Pr}(r(s_i) \ sees \ q)) = \sum_{j=1}^{z} p_{i,j} \mathbf{Pr}(r(s_{i,j}) \ sees \ q).$$

فرض کنید $s_{k,1}, s_{k,2}, ..., s_{k,l}$ مکان‌های ممکن s_k در $\mathcal{D}_k$ باشد که $\overline{r(s_{i,j})q}$ را قطع می‌کنند. احتمال اینکه s_k، $\overline{r(s_{i,j})q}$ را قطع نکند مساوی $p_k^{i,j} = (1 - p_{k,1} - p_{k,2} - ... - p_{k,l})$ است.

$$\mathbf{Pr}(q \ sees \ r(s_i)) = \sum_{j=1}^{z} p_{i,j} p_1^{i,j} . p_2^{i,j} .... p_n^{i,j}.$$

تعداد مکان‌های ممکن برای نقاط انتهایی $۲nz$ است و $P(q\ sees\ r(s_i))$ را می‌توانیم در $O(zn)$ به‌دست آوریم. بنابراین $E(ve(q))$ در زمان $O(n^۲z^۲)$ محاسبه می‌شود.

برای حالت $z = ۲$ الگوریتم سریعتری را بیان می‌کنیم. فرض کنید $a \in s_i$ یک نقطه‌ی انتهایی s_i و همچنین $s'_۱, s'_۲, ..., s'_k$ مجموعه‌ی پاره‌خط‌هایی باشد که $\overline{aq}$ را قطع می‌کنند. از آنجایی که احتمال انتخاب پاره‌خط‌ها مستقل از یکدیگر هستند خواهیم داشت:

$$\mathbf{Pr}(q\ sees\ a) = p_i.(۱ - p'_۱)(۱ - p'_۲)...(۱ - p'_k).$$

که نتیجه می‌دهد:

$$E(ve_p) = \sum_{a \in s_i} \mathbf{Pr}(q\ sees\ a).$$

بنابراین برای هر نقطه‌ی انتهایی، پاره‌خط‌هایی که $\overline{aq}$ را قطع می‌کنند را لازم داریم. با استفاده از قضیه ۵.۲ اگر $n \le k \le n^۲$، می‌توانیم در زمان $O_\epsilon(k)$ بر روی پاره‌خط‌ها پیش‌پردازش انجام دهیم به‌طوریکه برای هر پاره‌خط داده‌شده پاره‌خط‌هایی که آن را قطع می‌کنند در زمان $O_\epsilon(n/\sqrt{k})$ به‌دست آوریم. با استفاده از قضیه ۵.۲ می‌توانیم $\mathbf{Pr}(q\ sees\ a)$ را در زمان $O(n/\sqrt{k})$ به‌دست آوریم. پس برای $۲n$ نقطه‌ی انتهایی، $E(ve_p)$ را می‌توانیم در زمان $n.O(n/\sqrt{k})$ محاسبه کنیم. اگر قرار دهیم $k = n^{\frac{۴}{۳}}$ قضیه زیر نتیجه می‌شود.

قضیه ۶.۶ *مسأله‌ی شمارش احتمالاتی را در حالت $z = ۲$ می‌توان با ضریب تقریب ۲ در زمان $O(n^{\frac{۴}{۳}})$ پاسخ داد.*

۳.۶ نتیجه‌گیری

در این فصل مدلی احتمالاتی از مسأله‌های دید و شمارش را تعریف کردیم و نتایج بدست آمده در حل مسأله‌ها برای این مدل را ارائه کردیم. در حالت کلی ثابت کردیم که مسأله‌ی دید $\#P - complete$ است. ولی برای حالتی که یک پاره‌خط ۲ مکان ممکن دارد و در هر حالت در یکی از مکان‌های ممکن قرار می‌گیرد سختی مسأله را بررسی نکرده‌ایم. توجه به این نکته ضروری است که مسأله‌ی شمارش تعداد تطابق‌های کامل برای گراف‌های ۲ بخشی ۳ منتظم نیز $\#P - complete$ است [۲۱] . بنابراین تنها حالتی که باید بررسی شود بدست آوردن سختی مسأله در حالت ۲ مکان است. همچنین در رابطه با سختی مسأله‌ی شمارش نیز چیزی نگفته‌ایم. این نیز از مسایلی است که می‌تواند در آینده بررسی شود. توجه کنید که $\#P - complete$ بودن مسأله‌ی شمارش احتمالاتی را نمی‌توان از $\#P - complete$

بودن مسأله‌ی دید احتمالاتی نتیجه گرفت. مسأله‌ی دید احتمالاتی در زمان بهینه حل شده است زیرا می‌توان مسأله‌ی مرتب‌سازی به این مسأله کاهش داد. از مسایل دیگری که در ادامه‌ی این پروژه می‌توان در نظر گرفت تلاش برای بهبود زمان اجرا با پیش‌پردازش است.

فصل ۷

نتیجه‌گیری و کارهای آینده

در فصل پایانی تحقیق خلاصه‌ای از نتایج بدست آمده را بیان می‌کنیم. سپس ایده‌های استفاده شده و مسایلی که در آینده می‌توان بررسی کرد را به‌طور کلی مطرح می‌کنیم.

۱.۷ خلاصه‌ای از نتایج بدست آمده

در جدول ۱.۷ خلاصه‌ای از نتایج به‌دست آمده برای حل مسأله‌ی شمارش بیان و همچنین نتایج به‌دست آمده با بهترین روش‌های موجود پیشین مقایسه شده‌است. همانطور که دیده می‌شود بین زمان پیش‌پردازش و حافظه و زمان پرس‌وجوی الگوریتم‌های بیان شده مصالحه وجود دارد. برای تعیین مقدار مناسب زمان پرس‌وجو باید به تعداد پرس‌وجوها و همین‌طور میزان حافظه‌ی در دسترس توجه کرد. برای مقایسه‌ی کارایی الگوریتم بیان شده در فصل ۳ توجه کنید که اگر میزان حافظه، k، را در[۳۵] مساوی $O(m^{۲-۳\beta/۲})$ در نظر بگیریم در این‌صورت زمان پرس‌وجو مساوی $O_\epsilon(m^{۳/۴})$ خواهد بود. درصورتی‌که زمان اجرای الگوریتم فصل ۳ با همان مقدار حافظه مساوی $O(۱/\delta^۳ m^{\beta/۲}\log m)$ خواهد بود که اگر δ را عدد ثابتی درنظر بگیریم زمان پرس‌وجو بهبود یافته است.

جدول ۱.۷: مقایسه‌ی نتایج ارائه شده در این تحقیق برای مسأله‌ی شمارش با روش‌های پیشین

مسأله‌ی شمارش	زمان ساخت	حافظه	زمان پرس‌وجو	ضریب تقریب
[۳۵]	$O_\epsilon(k)$	$O_\epsilon(k)$	$O_\epsilon(m/\sqrt{k})$	۲
[۴۵]	$O(k\log(\sqrt{k}/n))$	$O(k)$	$O(n^4\log(\sqrt{k}/n)/\sqrt{k})$	۲
فصل ۳	$O(m^{2-3\beta/2}\log m)$	$O(m^{2-3\beta/2})$	$O(1/\delta^3 m^{\beta/2}\log m)$	$1+\delta$
فصل ۴	$O(n^2)$	$O(n^2\log n)$	$O(1/\delta\sqrt{n}\log n)$	$1+\delta$
فصل ۴	$O_\epsilon(nk)$	$O_\epsilon(k)$	$O_\epsilon(n^2/\sqrt{k})$	مقدار دقیق

در ادامه، نتایجی هم در فضای ۳ بعدی بدست آورده بودیم که در جدول‌ها نشان داده شده‌اند. همچنین مدلی احتمالاتی از مسأله را تعریف کردیم و الگوریتمی با $O(n\log n)$ در حالت خاص برای مسأله‌ی دید احتمالاتی ارائه کردیم. برای حالت کلی نیز ثابت کردیم مسأله $\#P-complete$ است. همچنین روشی تقریبی برای مسأله‌ی شمارش احتمالاتی با زمان اجرای $O(n^{4/3})$ در حالت خاص و $O(z^2n^2)$ در حالت کلی ارائه کردیم.

۲.۷ ایده‌های استفاده شده

در این قسمت ایده‌های استفاده شده را بیان می‌کنیم.

- ارائه‌ی الگوریتم‌های تصادفی و تحلیل زمان و حافظه‌ی مورد استفاده.

استفاده از الگوریتم های تصادفی به همراه تحلیل درست امید ریاضی زمان و حافظه‌ی مورد استفاده معمولا از ابزارهایی است که برای بهبود پیچیدگی زمانی و حافظه‌ی مورد نیاز حل مسأله‌ها استفاده می‌شود. در این تحقیق نیز چندین الگوریتم تصادفی ارائه و تحلیل شده‌اند که به بهبود زمان و حافظه مورد استفاده کمک کرده‌اند.

- ارائه‌ی الگوریتم‌های تقریبی و تصادفی.

در برخی از مسایل بدست آوردن مقدار دقیق جواب مسأله مورد نیاز نیست. در این تحقیق نیز چندین الگوریتم تصادفی با زمان و حافظه‌ی بهبود یافته برای حل مسأله ارائه شده‌اند. تحلیل ضریب تقریب و محاسبه‌ی زمان و حافظه‌ی مورد نیاز نیز از مسایلی بوده است که بررسی شده‌اند.

- استفاده از ابزارهای ریاضی همانند نظریه گراف.

مدلسازی مسأله به صورت یک مسأله‌ی ریاضی و تلاش برای حل مسأله با استفاده از ابزارهای ریاضی همانند نظریه گراف از ایده‌هایی است که برای بهبود ضریب تقریب الگوریتم تصادفی مورد استفاده قرارگرفته است.

- ترکیب الگوریتم‌های پیشین با الگوریتم‌های جدید.

در برخی مواقع می‌توان با ترکیب الگوریتم‌ها به زمان و حافظه یا ضریب تقریب بهتری دست یافت. به عنوان مثال در برخی از الگوریتم‌های ارائه شده در ۲ مرحله جواب را بدست می‌آوریم. در قسمت اول تا زمان مشخصی از الگوریتم‌های پیشین استفاده می‌کنیم. چنانچه به جواب رسیدیم الگوریتم متوقف می‌شود. در غیراین‌صورت از الگوریتم ارائه شده در مرحله‌ی دوم استفاده می‌کنیم. اگر تا زمان مشخص به جواب برسیم مقدار دقیق مسأله در زمان دلخواه بدست می‌آید. در غیر این‌صورت ثابت می‌شود که جواب بدست آمده در مرحله‌ی دوم یک مقدار تقریبی از جواب اصلی است. برای تحلیل ضریب تقریب از این نکته استفاده می‌شود که مقدار جواب مسأله بزرگتر از یک مقدار خاص است.

- اجرای الگوریتم‌ها بر روی داده‌های واقعی.

بیشتر الگوریتم‌های تصادفی را می‌توان به سادگی پیاده‌سازی کرد. تحلیل زمان و حافظه‌ی مورد نیاز از سختی‌های این الگوریتم ها است. در این تحقیق نیز برخی از الگوریتم‌ها را برروی داده‌های واقعی اجرا کرده‌ایم. از این نتایج برای اثبات مجدد الگوریتم‌های ارائه شده استفاده کرده‌ایم (هرچند که اثبات ریاضی نیز داشته‌اند). در برخی موارد نیز که موفق به تحلیل ریاضی الگوریتم‌های تصادفی و تقریبی نشده‌ایم، نتایج تجربی به‌دست آمده نشان داده‌اند که الگوریتم‌های ارائه شده در زمان مناسب اجرا می‌شوند و خطای قابل قبولی دارند.

۳.۷ پیشنهادها برای ادامه‌ی کار

در این بخش مباحثی که در ادامه‌ی این پروژه می‌توان بررسی کرد را به‌طور کلی بیان می‌کنیم.

۱.۳.۷ ادامه‌ی پروژه براساس تکمیل و بهبود نتایج به‌دست آمده

موضوع کلی که به‌عنوان رساله‌ی دکتری در نظر گرفته شده‌است مسأله‌ی شمارش اشیا دیدپذیر و تعیین دیدپذیر بود اشیا نسبت به هم است که به‌صورت دقیق در فصل ۱ تعریف شده‌اند. نتایج موجود و نتایج جدیدی را که به‌دست آورده‌ایم، بیان کردیم. در هر فصل روش‌ها و ایده‌های جدیدی را که به‌کار برده‌ایم توضیح دادیم. مسایل جزیی‌تری را که در ادامه می‌توان بر روی آنها تمرکز کرد، به‌طور کامل در انتهای هر فصل توضیح داده‌ایم. بررسی و گسترش این روش‌ها در ادامه‌ی این تحقیق مدنظر قرار خواهدگرفت.

در فصل ۳ مسأله‌ی ۵.۱ را به‌صورت یک مسأله در نظریه‌ی گراف مدل‌سازی کردیم. اثبات کردیم که حل مسأله‌ی شمارش معادل با به‌دست آوردن تعداد مولفه‌های همبندی گراف خواهدبود. روشی تصادفی برای به‌دست آوردن مقدار تقریبی تعداد مولفه‌ها ارائه دادیم. در ادامه، هدف ارائه‌ی داده‌ساختاری است که با زمان موردنظر ما بتواند تعداد مولفه‌ها را به‌دست آورد.

همچنین گفتیم که در [۳۵] توانسته‌اند منطقه‌ی دیدپذیر هر پاره‌خط را به‌صورت اجتماعی از مثلث‌ها به‌دست آورند. سپس با انجام پیش‌پردازش بر روی این مثلث‌ها با استفاده از درخت افراز، در مرحله‌ی پرس‌وجو به‌ازای نقطه‌ی پرس‌وجوی p تعداد مثلث‌های شامل p را به‌دست می‌آورند. از آنجایی که مثلث‌های مربوط به یک پاره‌خط ممکن است باهم اشتراک داشته‌باشند با این روش برخی از پاره‌خط‌ها چندین بار شمرده می‌شوند. آنها ثابت کرده‌اند که تعداد مثلث‌های شامل نقطه‌ی p حداکثر ۲ برابر تعداد پاره‌خط‌های دیدپذیر است. در ادامه‌ی این تحقیق مسأله را به این‌صورت در نظر می‌گیریم

سؤال ۱.۷ *می‌خواهیم برروی تعدادی مثلث که هرکدام دارای یک برچسب هستند و این برچسب‌ها ممکن است تکراری باشند، پیش‌پردازش انجام دهیم به‌طوری که به‌ازای نقطه‌ی پرس‌وجوی p، تعداد برچسب‌های مجزا را که به مثلث‌های شامل p نسبت داده‌شده‌اند، به‌دست آوریم.*

مسایل مشابه بسیاری در این زمینه وجود دارد که حل این مسأله منجر به حل این مسایل نیز خواهد شد (مراجعه شود به [۳۷]). در فصل ۴ مسأله را در حالت خاص مربوط به مسأله‌ی خودمان بررسی و حل کردیم.

۲.۳.۷ حل مسأله در حالت‌های خاص

در ادامه‌ی این تحقیق حالت‌های مختلف مسأله را نیز می‌توان در نظر گرفت. به‌عنوان مثال حالتی که در آن به جای یک نقطه‌ی پرس‌وجو، پاره‌خط وجود داشته باشد و یا نقطه پرس‌وجو به‌صورت متحرک

در نظر گرفته شود. هدف از این کار اعمال ایده‌های استفاده شده بر روی حالت‌های مختلف است. چون در برخی از موارد ایده‌های استفاده شده را می‌توان گسترش داد و برای حالت‌های دیگر مسأله و مسایل مشابه دیگر نیز استفاده کرد.

تعریف‌های مختلفی برای دیدپذیری در مقالات مختلف با مفاهیم گوناگون وجود دارد که حل مسایل با توجه به آن تعریف‌ها سختی‌های خاص خود را دارد. در ادامه می‌توان بررسی کرد که با استفاده از روش‌های جدیدی که در این گزارش آمده‌است آیا می‌توان راه‌حل‌های ساده‌تر و بهینه‌تری برای حل مسایل مشابه با تعریف‌های گوناگون ارائه داد؟

همچنین بررسی مسأله در حالتی که ورودی مسأله به جای پاره‌خط‌ها اشیایی با خصوصیات مشخص شده باشند نیز مدنظر قرارخواهد گرفت. به عنوان مثال همان‌طور که در فصل ۲ گفته شد، در [۲۸] ثابت شده‌است که با اعمال شرایط خاصی بر روی ورودی، به صورت قابل ملاحظه‌ای از پیچیدگی مسأله کاسته می‌شود. از آن‌جایی که در عمل در برخی حالت‌ها، ورودی مسأله‌ها دارای این شرایط خاص هستند، تمرکز بر روی این حالت‌ها قابل توجیه است.

۳.۳.۷ بررسی مسأله در فضاهای با بعد بزرگ‌تر از ۲

مسایل مربوط به دیدپذیری در فضاهای با بعد بزرگتر از ۲ کاربرد و اهمیت بسیاری دارند. در فصل ۲ تعدادی از نتایج به دست آمده توسط افراد مختلف عنوان شد. در ادامه‌ی این تحقیق می‌توان مسایل در فضاهای با بعد بزرگتر از ۲ را نیز بررسی کرد. چون ایده‌ها و روش‌های ارائه شده در فضای ۲ بعد در برخی مواقع قابل گسترش به فضاهای با بعد بزرگتر از ۲ نیز هستند. به عنوان مثال همان‌طور که در فصل ۵ بیان شد، توانستیم قضیه‌ای را که در فضای ۲ بعدی ارائه شده بود به فضای ۳ بعدی تعمیم دهیم. از آنجایی که پیچیدگی مسأله‌ها در این حالت‌ها بیشتر می‌شود، ما به دنبال طراحی روش‌ها و الگوریتم‌هایی با زمان و حافظه‌ی مناسب خواهیم بود.

۴.۳.۷ مدل احتمالاتی

تعریف مدل احتمالی مسأله نیز در این تحقیق مورد بررسی قرار گرفت و الگوریتم‌هایی برای حل این مسایل مطرح شد. مسایلی که حل نشده بودند نیز عنوان شدند. در ادامه‌ی این تحقیق این مسایل حل نشده را نیز می‌توان بررسی کرد. از جمله این مسایل بررسی پیچیدگی محاسبه‌ی مسأله‌ی دید احتمالاتی در حالت ۲ مکان ممکن بود که توضیح دادیم. همچنین بررسی پیچیدگی مسأله‌ی شمارش احتمالاتی نیز از جمله مسایل باز است. مسأله‌ی دیگری که می‌توان به آن پرداخت تلاش برای بهبود زمان اجرای

الگوریتم‌های ارائه شده و ارائه‌ی روش‌هایی برای ایجاد مصالحه بین زمان اجرا و حافظه در حالت احتمالاتی است.

فهرست مراجع

[1] Abam, M., Alipour, S., Ghodsi, M., Mahdian, M.: Visibility Testing and Counting for Uncertain Segments EUROCG2016, (2016) 9

[2] Agarwal, P.K., Sharir, M.: Applications of a new space-partitioning technique. Discrete and Computational Geometry 9, 9–11 (1993) 15

[3] Agarwal, P.K., Kreveld, J.v.K:Connected Component and Simple Polygon Intersection Searching. Algorithmica 15(6): 626–660 (1996) 18

[4] Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In Chazelle, B., Goodman, J.E., Pollack, R. (eds) Advances in Discrete and Computational Geometry, Contemporary Mathematics, vol. 233, pp. 1–56. American Mathematical Society Press, Providence (1999) 15

[5] Agarwal, P.K.: Partitioning arrangements of lines, part I: An efficient deterministicalgorithm. Discrete and Computational Geometry, 5, 449–483 (1990) 18

[6] Agarwal, p.k.: Range searching. In J. E. Goodman and J. O'Rourke, editors, Handbook of Discrete and Computational Geometry, 2nd edn., chapter 36, Chapman and Hall, CRC (2004) 15

[7] Agarwal, P.K.: Ray shooting and other applications of spanning trees with law stabbing number. SIAM. Comput, 21(3), 540-570 (1992) 18

[8] Alipour, S., Ghodsi, M. and Jafari, A.: An Improved Constant-Factor Approximation Algorithm for Planar Visibility Counting Problem. COCOON 2016: 209-221(2016) 8, 9

[9] Alipour, S., Ghodsi, M., Zarei, A., and Pourreza, M.:Visibility testing and counting Inf. Process. Lett. 115(9): 649-654 (2015) 8, 9

[10] Alipour, S., Ghodsi, M., Gudukbay, U. and Golkari, M.:An Approximation Algorithm for Computing the Visibility Region of a Point on a Terrain and Visibility Testing VISAPP (3) 2014: 699-704 (2014) 9

[11] Alipour, S., Zarei, A: Visibility testing and counting. Frontiers in Algorithmics and Algorithmic Aspects in Information and Management Lecture Notes in Computer Science, 6681, 343–351 (2011) 13, 34, 42, 56

[12] Aronov, B., Guibas, L.J., Teichmann, M., Zhang, L.: Visibility queries and maintenance in simple polygons. Discrete and Computational Geometry 27, 461–483 (2002) 11, 12

[13] Aronov, B., Guibas, L.J, Teichmann, M. and Zhang, L.: Visibility queries in simple polygons and applications. In: International Symposium on Algorithms and Computation, pp. 357–366 (1998) 11

[14] Asano, T.: An efficient algorithm for finding the visibility polygon for a polygonal region with holes. IEICE Transactions, 557–589 (1985) 3

[15] Asano, T., Guibas, L.J., Hershberger, J., Imai, H.: Visibility of disjoint polygons. Algorithmica 1(1), 49–63 (1986) 12

[16] Ben-Moshe, B., Carmi, P., Katz, M. J.: Approximating the visible region of a point on a terrain. Proc. algorithm engineering and experiments (ALENEX'04), 120–128 (2004) 13

[17] Bondy, J. A., Murty, U. S. R.: Graph theory with applications (Vol. 290). London: Macmillan (1976) 21

[18] Bose, P., Lubiw, A., Munro, J.I.: Eficient visibility queries in simple polygons. Computational Geometry Theory and Applications 23(7), 313–335 (2002) 11

[19] Chen, D., Wang, H.:Visibility and ray shooting queries in polygonal domains In F. Dehne, R. Solid-Oba, J.-R. Sack(Eds.) Algorithms and data structures, Vol. 8037 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 244-255 (2013) 11

[20] Cole, R., Sharir, M.:Visibility problems for polyhedral terrains. Journal of Symbolic Computation 7(1), 11–30 (1989) 43

[21] Dagum, P., Luby, M., Mihail, M., and Vazirani, U.: Polytopes, permanents and graphs with large factors. In Foundations of Computer Science, 29th Annual Symposium on (pp. 412-421). IEEE.(1988) 57

[22] Devai, F.:Quadratic bounds for hidden line elimination Proceedings of the second annual symposium on Computational geometry, 269–275, 1986 43

[23] Chan, T. M.: Optimal partition trees. In: SoCG '10 Proceedings of the 2010 annual symposium on Computational geometry, pp. 1–10 (2010) 16

[24] Chazelle, B., Guibas, L.J., Lee, D. T.: The power of geometric duality. BIT Numerical mathematics 25(1), 76–90 (1985) 18

[25] Chazelle, B., Welz, E.: Quise-optimal range rearching in space of finite VC-Dimention. Discrete and Computational Geometry 4, 467–489 (1989) 15

[26] Chazelle, B., Sharir, M., Welz, e.: Quasi-optimal upper bounds for simplex range searching and new zone theorems In: SCG '90 Proceedings of the sixth annual symposium on Computational geometry, pp.23-33 (1990) 16

[27] De Berg, M. O., Cheong, M., Kreveld, V. and Overmars, M.: Computational Geometry Algorithms and Applications. Springer, third edition, (2008) , 14, 17

[28] De Berg, M., Haverkort, H., Tsirogiannis, C. P.: Visibility maps of realistic terrains have linear smoothed complexity. In Proceedings of the 25th annual symposium on Computational geometry ACM, pp. 163–168 (2009) 13, 63

[29] Erickson, J.: New lower bounds for Hopcroft's problem. Discrete and Computational Geometry 16(4), 389–418 (1996) 41

[30] Fischer, M., Hilbig, M., Jahn, C., auf der Heide, F.M., Ziegler, M.: Planar visibility counting. CoRR, abs/0810.0052 (2008) , 1, 5, 12

[31] Fischer, M., Hilbig, M., Jahn, C., auf der Heide, F.M., Ziegler, M.: Planar visibility counting. In: Proceedings of the 25th European Workshop on Computational Geometry(EuroCG 2009), pp. 203–206 (2009) 12

[32] Ghosh, S. K. and Mount, D.: An output sensitive algorithm for computing visibility graphs. SIAM Journal on Computing 20, 888–910 (1991) 4, 11

[33] Ghosh, S. K.: Visibility algorithms in the plane. Cambridge university press, Cambridge (2007) 11

[34] Gigus, Z., Canny, J., and Seidel, R.: Efficiently computing and representing aspect graphs of polyhedral objects. IEEE Trans. Pattern Anal. Mach. Intell., 13(6), 542–-551 (1991) 7

[35] Gudmundsson, J., Morin, P.: Planar visibility: testing and counting. In: Annual Symposium on Computational Geometry, pp. 77–86 (2010) , 8, 12, 13, 14, 24, 29, 34, 38, 40, 41, 59, 60, 62, 69

[36] Heffernan, P.J., Mitchell, J. S. B.: An optimal algorithm for computing visibility in the plane. SIAM J. Comput. 24(1), 184–201 (1995)

[37] Janardan, R., Lopez, M.: Generalized intersection searching problems. International Journal of Computational Geometry and Applications 3.01 39-69 (1993) 38, 62

[38] LaValle, S.M.: Planning Algorithms. Cambridge University Press, (2006) 7

[39] Matoutsek, J.: Construction of ϵ-nets. Discrete and Computational Geometry 5, 427–448 (1990) 18

[40] Matoutsek, J.: Effcient partition trees. Discrete and Computational Geometry 8, 315–334 (1992) 16, 17

[41] Matoutsek, J.:Cutting hyperplane arrangements. Discrete and Computational Geometry 6, 385–406 (1991) 16

[42] Moet, E., Knauer, C., Van Kreveld, M.:Visibility maps of segments and triangles in 3d. Computational Science and Its Applications-ICCSA 2006, 20–29 (2006) 13

[43] Morini, E., Rocchi, F., Avizzano, A. C., Bergamasco, M.: Visibility Techniques Applied to Robotics. In: 19th IEEE International Symposium on Robot and Human Interactive Communication Principe di Piemonte - Viareggio, Italy, Sept, pp. 12–15, (2010) 1

[44] Ben-Moshe, .B, Carmi, P. and Katz, M. J.:Approximating the Visible Region of a Point on a Terrain Geoinformatica, 12(1), 21–36 (2008) , 43, 48, 49, 50

[45] Nouri, M. and Ghodsi, M.: Space/query-time tradeoff for computing the visibility polygon. Computational Geometry. 46(3), 371–381 (2013) 12, 60

[46] Nouri, M, Daneshpajouh, S., Alipour, S. and Ghodsi, M.:Weak visibility counting in simple polygons J. Computational Applied Mathematics 288: 215-222 (2015) 9

[47] Pocchiola, M., Vegter, G.: The visibility complex. International Journal of Computational Geometry and Applications 6(3), 279–308 (1996) 11, 12

[48] Richard, C., Sharir, M.: Visibility problems for polyhedral terrains. Journal of Symbolic Computation 7(1), 11–30 (1989) 13

[49] Riegler, H.: Point-visibility in computer games. Computer Graphics Seminar. Computer Game Technology, (2001) 1, 7

[50] Stiene, S., Hertzberg, J.: Virtual range scan for avoiding 3D obstacles using 2D tools. Advanced Robotics, 2009. ICAR 2009. International Conference on. IEEE, (2009) 7

[51] Suri, S., O'Rourke, J.: Worst-case optimal algorithms for constructing visibility polygons with holes. In: Proceedings of the Second Annual Symposium on Computational Geometry (SCG 84), pp. 14–23 (1984) 3

[52] Valiant, L. G. The complexity of computing the permanent. *Theoretical computer science*8(2):189–201 ,1979. 53

[53] Vegter, G.: The visibility diagram: A data structure for visibility problems and motion planning. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT(1990). LNCS, 447, pp. 97–110. Springer, Heidelberg (1990) 12, 24, 26, 31, 34, 35

[54] Wang, C. A., Zhu, B.: Three-dimensional weak visibility: Complexity and applications. Theoretical computer science, 234(1), 219–232, (2000) 13

[55] Welz, E.:Partition trees for triangle counting and other range searching problems. In: SCG '88 Proceedings of the fourth annual symposium on Computational geometry, pp. 22–33 (1988) 16

[56] Zarei, A., Ghodsi, M.: Efficient computation of query point visibility in polygons with holes. In: Proceedings of the 21st Annual ACM Symposium on Computational Geometry(SCG 2005) (2005) 11, 12

Abstract

Planar visibility computing is defined as determining the region of the plane that is visible from a specific observer. This concept has many applications in computer graphics, robotic and computer games. In certain visibility problems, counting the number of visible objects in an appropriate time is required. For obtaining a solution fast, current algorithms give an approximated count. In this thesis, we consider visibility testing problem and visibility counting problem.

For a given set $S = \{s_1, s_2, ..., s_n\}$ of non-intersecting segments and a query point p in the plane, the visibility testing problem checks the inter-visibility of p and a segment $s_i \in S$ and the visibility counting problem counts the number of visible segments from p. We have introduced two approximation algorithms. In the first algorithm, we have a trade-off between space and query time and also a trade-off between approximation factor and query time. In the second algorithm, we have used random sampling method. The expected query time of this algorithm is better than the first algorithm in certain cases. Next, we have proposed a randomized algorithm that gives the exact solution for the visibility counting problem with the same time and space complexity of [35]. Then, we have considered the problems in 3D and have given algorithms to solve them. We have implemented these algorithms on real data sets. The results show that the time complexity and approximation factor of the proposed algorithms are effective and applicable in practice. At last, we have introduced a probabilistic model for these problems and proposed some solutions for them. In certain cases, we have proved that the probabilistic visibility testing problem is $\#P - complete$. Also, we have presented some approximation algorithms to solve the probabilistic visibility counting problem.

Keywords: 1-Computational geometry, 2-Visibility, 3-Approximation algorithms, 4-Randomized algorithms

Sharif University of Technology
Computer engineering department

PhD Thesis

Topic

Efficient algorithms for visibility testing of objects and counting

By
Sharareh Alipour

Supervisor
Mohammad Ghodsi

Sep 2016