پرس و جوی موازی کوتاه‌ترین مسیر روی سطوح چندوجهی وزن‌دار

حمید ضرّابی‌زاده
zarrabi@basu.ac.ir
دانشکده‌ی مهندسی کامپیوتر
دانشگاه بوعلی سینا

محمّد قدسی
ghodsi@sharif.edu
دانشکده‌ی مهندسی کامپیوتر
دانشگاه صنعتی شریف

چکیده

مسئله‌ی پیدا کردن کوتاه‌ترین مسیر یکی از مسئله‌های مهمّ مطرح شده در زمینه‌ی هندسه‌ی محاسباتی و شاخه‌های دیگری هم‌چون الگوریتم‌های گراف، سیستم‌های اطّلاعات جغرافیایی (GIS)، روباتیک و غیره می‌باشد. مسئله‌ی موردنظر ما حالت خاصّی از این مسئله است که هدف آن تعیین کوتاه‌ترین مسیر بین دو نقطه از یک سطح وزن‌دار در فضای سه بعدی است، به گونه‌ای که تمام مسیر روی سطح یک چندوجهی قرار گرفته باشد. در این مقاله پس از معرّفی دقیق مسئله و بررسی کارهای انجام شده در این زمینه، یک الگوریتم تقسیم و حل برای پرس و جوی کوتاه‌ترین مسیر در سطوح نامنظم مثلث بندی شده‌ی وزن دار معرّفی می کنیم و براساس آن، یک الگوریتم موازی برای مسئله ارائه خواهیم کرد. سپس با بررسی نتایج به‌دست آمده از پیاده‌سازی این الگوریتم موازی، به مقایسه‌ی میزان تسریع الگوریتم موازی نسبت به الگوریتم ترتیبی خواهیم پرداخت.

کلیدواژه‌ها: هندسه‌ی محاسباتی، الگوریتم موازی، الگوریتم تقریبی، کوتاه‌ترین مسیر، چندوجهی وزن‌دار.

۱ مقدّمه

تحقیقات اخیر در زمینه‌ی روباتیک و پیمایش زمین باعث گسترش تحقیقات در زمینه‌ی برنامه‌ریزی حرکت شده است. یکی از عمده‌ترین مسائل در این زمینه، مسائل مربوط به پیدا کردن کوتاه‌ترین مسیر است که از مسائل پایه‌ای در هندسه‌ی محاسباتی و نظریّه‌ی گراف‌ها محسوب می‌شود.

در این مقاله، به نوع خاصّی از این مسئله توجّه خواهیم کرد که هدف آن تعیین کوتاه‌ترین مسیر بین دو نقطه از یک چندوجهی وزن‌دار در فضای سه بعدی است. چندوجهی وزن‌دار، سطحی است که به هر وجه آن یک عدد حقیقی نامنفی نسبت داده شده است که نشان دهنده‌ی هزینه‌ی عبور از واحد طول در آن وجه است. برای حالت بدون وزن این مسئله الگوریتم‌های با زمان اجرای از مرتبه‌ی چندجمله‌ای وجود دارد ولی در حالت کلّی و در فضای سه بعدی که در آن تعدادی مانع به شکل چندوجهی وجود دارند، این مسئله یک مسئله‌ی NP-Hard محسوب می‌شود.

در این مقاله ضمن بررسی کارهای انجام شده، به معرّفی یک الگوریتم تقسیم و حل برای پرس و جوی کوتاه‌ترین مسیر روی سطح یک چندوجهی وزن‌دار می‌پردازیم و با استفاده از آن، یک الگوریتم موازی برای حلّ مسئله ارائه می‌دهیم. سپس با بررسی نتایج عملی حاصل از پیاده‌سازی الگوریتم موازی، میزان تسریع الگوریتم را، هم از لحاظ عملی و هم از جنبه‌ی نظری بررسی و تحلیل خواهیم کرد.

Algorithm	Running Time
Mitchell and Papadimitriou	$O(n^{8}\log(\frac{nK}{N}))$
Lanthier et al.	$O(n^{5})$
Mata and Mitchell	$O(n^{3}K)$
Aleksandrov et al.	$O(n(K\log K)\log(nK\log K))$

جدول ۱: الگوریتم‌های ترتیبی یافتن کوتاه‌ترین مسیر روی سطوح وزن‌دار

۲ تعاریف اولیه

چندوجهیِ وزن‌دار P به‌همراه دو نقطه‌ی s و t (نقاط مبداً و مقصد) روی سطح چندوجهی داده شده است؛ هدف پیدا کردن کوتاه‌ترین مسیر وزن‌دار بین این دو نقطه روی سطح چندوجهی می‌باشد. بدون این‌که از کلّیت مسئله کاسته شود، می‌توانیم فرض کنیم که تمامی وجه‌ها به‌شکل مثلّث‌اند و نقاط s و t دو رأس از این مثلّث‌بندی هستند.

به هر وجه f از P، یک عدد مثبتِ w_f به‌عنوان وزنِ آن وجه اختصاص داده شده است. در حقیقت این عدد برابر هزینه‌ی عبور از واحد طول در وجه می‌باشد. اگر فاصله‌ی اقلیدسی بین دو نقطه‌ی x و y را با $|xy|$ نمایش دهیم، فاصله‌ی وزن‌دار آن‌ها در یک وجه به وزن w_f برابر $w_f|xy|$ خواهد بود که این فاصله‌ی وزن‌دار را به‌صورت $||xy||$ نشان خواهیم داد. همچنین، کوتاه‌ترین مسیر وزن‌دار بین نقاط s و t را با $\Pi(s,t)$ و طول آن را با $||\Pi(s,t)||$ نمایش خواهیم داد.

۳ بررسی کارهای انجام‌شده

در این بخش به بررسی الگوریتم‌هایی که تا کنون برای یافتن کوتاه‌ترین مسیر روی سطوح وزن‌دار، چه در حالت ترتیبی و چه در حالت موازی ارائه شده‌اند، می‌پردازیم.

۱.۳ الگوریتم‌های ترتیبی

با این که برای یافتن کوتاه‌ترین مسیر در سطوح وزن‌دار الگوریتم دقیقی وجود ندارد، در عوض الگوریتم‌های تقریبی گوناگونی برای این مسئله ارائه شده‌اند.

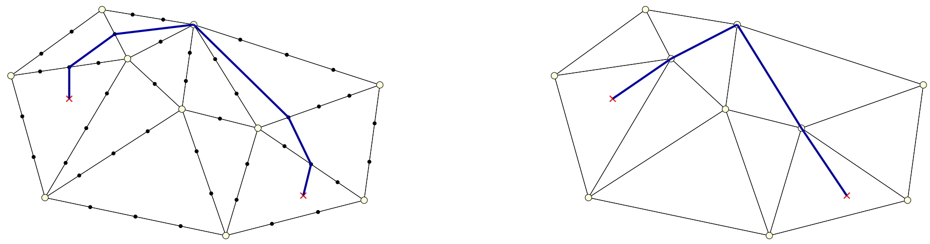
در جدول ۱، الگوریتم‌های تقریبی موجود به‌همراه زمان اجرایی هر یک نشان داده شده است. برای این‌که بتوان زمان اجرای این الگوریتم‌ها را به‌راحتی با یک‌دیگر مقایسه نمود، تمامی زمان‌های اجرا در این جدول بر اساس یک پارامتر $K=O(\frac{N^{2}W}{\epsilon w})$ مشخص شده‌اند. در این پارامتر، N برابر بزرگ‌ترین مختصات صحیح نقاط سطح، W و w به‌ترتیب برابر بیش‌ترین و کم‌ترین وزن صحیح ناحیه‌ها و ϵ درجه‌ی دقّتِ تعریف‌شده توسّط کاربر است. جزئیات بیش‌تر این الگوریتم‌ها در مراجع [۹، ۱۰] آمده است.

همان‌طور که در جدول ۱ مشاهده می‌شود، الگوریتم Aleksandrov و همکارانش توانسته است وابستگیِ زمان اجرا به n را نسبت به الگوریتم‌های دیگر به‌طور قابل ملاحظه‌ای کاهش دهد. با توجّه به این مطلب، ما نیز برای پیاده‌سازی الگوریتم موازی خود در بخش ۵ از الگوریتم Aleksandrov استفاده خواهیم کرد.

۲.۳ الگوریتم‌های موازی

الگوریتم‌های موازی مختلفی برای یافتن کوتاه‌ترین مسیر روی گراف‌ها ارائه شده است. به‌عنوان مثال، Leighton در مرجع [۷] نشان داده است که کوتاه‌ترین مسیر بین هر دو رأس[۱] از یک گراف وزن‌دار را می‌توان به‌وسیله‌ی الگوریتمِ بستار تعدّی گراف به‌طور موازی

[۱] All-Pairs Shortest Path



(ب) (الف)

شکل ۱: (الف) گراف تقریب ساده (ب) گراف تقریب با نقاط کمکی

محاسبه نمود. در این الگوریتم، کوتاه‌ترین مسیر بین هر دو رأس از یک گراف وزن‌دارِ n رأسی، توسّط یک ساختار توری‌شکلِ از پردازنده‌ها، با به‌کارگیری $O(n^{۲})$ پردازنده و در زمان $O(n)$ محاسبه می‌شود.

مسئله‌ی کوتاه‌ترین مسیر از یک رأسِ مبدأ به سایر رأس‌ها[۲] نیز از دیدگاه موازی‌سازی مورد بررسی قرار گرفته است. Paige و Kruskal در مرجع [۸] الگوریتمی را معرّفی کرده‌اند که در زمان $O(n\log n)$ و با استفاده از $O(n)$ پردازنده کوتاه‌ترین مسیر بین یک رأس مبدأ تا سایر رأس‌ها را محاسبه می‌کند. Cohen [۴] نیز الگوریتمی را معرّفی کرده است که کوتاه‌ترین مسیرها در یک گراف جهت‌دار را با استفاده از تجزیه‌های k^{μ}—جداساز[۳] محاسبه می‌کند. این الگوریتم کوتاه‌ترین مسیر بین s رأس مبدأ تا تمام رأس‌های دیگر را با مجموع کار $O(n^{۳\mu}+s(n+n^{۲\mu}))$ محاسبه می‌کند. با قرار دادن $\mu=۰/۵$ می‌توان از این کران برای گراف‌های مسطّح استفاده نمود. به‌طور کلّی کوتاه‌ترین مسیر بین s رأس مبدأ تا تمام رأس‌های دیگر با مجموع کار $O(M(n^{\mu})\log^{۲} n+s(n+n^{۲\mu}))$ قابل محاسبه است، که $M(k)=o(k^{۲/۳۷})$ بهترین کران شناخته شده برای ضرب ماتریس‌های $k\times k$ است.

بنا بر بررسی‌ها و جست‌وجوهای انجام‌شده، تا کنون هیچ الگوریتم موازی برای پیدا کردن کوتاه‌ترین مسیر روی سطح یک چندوجهی وزن‌دار با نقاط مبدأ و مقصدِ نامعیّن ارائه نشده است، امّا در مرجع [۲] یک الگوریتم تقسیم و حل برای پرس و جوی کوتاه‌ترین مسیر بین هر دو نقطه‌ی دل‌خواه از یک گراف ارائه شده است که ما نیز با استفاده از این الگوریتم تقسیم و حل، یک الگوریتم موازی برای انجام پرس‌وجوی کوتاه‌ترین مسیر روی سطوح چندوجهی وزن‌دار ارائه خواهیم کرد.

۴ پرس‌وجوی کوتاه‌ترین مسیر

در این بخش به معرفی یک الگوریتم تقسیم و حل می‌پردازیم که با پیش پردازش اطلاعات اولیه، داده‌ساختاری ایجاد می‌کنند که توسط آن می‌توان کوتاه‌ترین مسیر تقریبی بین هر دو رأس دل‌خواه از یک گراف را به‌دست آورد. این الگوریتم در مرجع [۲] معرفی شده است و ما نیز از این روش برای پیاده‌سازی الگوریتم موازی در بخش ۵ بهره خواهیم جست.

برای این که بتوانیم از الگوریتم‌های کوتاه‌ترین مسیر در گراف‌ها، برای چندوجهی‌های وزن‌دار استفاده کنیم، باید ابتدا چندوجهی را با یک گراف وزن‌دار تقریب بزنیم و سپس در گرافِ تقریب به‌دست آمده، کوتاه‌ترین مسیر بین نقاط مبدأ و مقصد را توسط الگوریتم‌های موجود در گراف‌ها محاسبه نماییم (شکل ۱—الف). با افزودن چند نقطه‌ی کمکی روی یال‌های گراف تقریب و ساختن یک گراف کامل روی نقاط کمکی مربوط به هر وجه گراف، می‌توان مسیر تقریبی به‌دست‌آمده را به کوتاه‌ترین مسیر واقعی نزدیک‌تر کرد (شکل ۱—ب). Aleksandrov و همکارانش [۱] توانسته‌اند با افزودن به‌طور متوسط m نقطه روی هر یال از گراف تقریب، یک الگوریتم ϵ—تقریبی برای یافتن کوتاه‌ترین مسیر روی سطوح چندوجهی وزن‌دار ارائه نمایند.

قضیّه‌ی ۱ . فرض کنید P یک nوجهی و s و t دو رأس از آن باشند. با صرف هزینه‌ی زمانی $O(mn\log mn)$ می‌توان کوتاه‌ترین مسیر وزن‌دارِ $\Pi(s,t)$ بین s و t را به‌وسیله‌ی مسیر $\Pi'(s,t)$ طوری تقریب زد که $\|\Pi'(s,t)\|\leq(۱+\epsilon)\|\Pi(s,t)\|$. (منظور از m، متوسط تعداد نقاط کمکیِ اضافه‌شده روی یال‌های گراف تقریب است.) [۱]

[۲] Single-Source Shortest Path

[۳] k^{μ}-Separator

نحوه‌ی پیاده‌سازی الگوریتم تقریبی مذکور، در مرجع [۹] به‌طور کامل بررسی شده است. قضایای زیر نیز در ادامه‌ی این بخش مورد استفاده قرار خواهند گرفت.

قضیّه‌ی ۲ . گراف مسطّحِ G با n رأس را می‌توان با صرف هزینه‌ی زمانی $O(n)$ به $O(n/r)$ ناحیه چنان افراز کرد که هر ناحیه شامل $O(r)$ رأس درون و $O(\sqrt{n})$ رأس روی مرز خود باشد. [۳]

قضیّه‌ی ۳ . در هر گراف G با n رأس و m یال، کوتاه‌ترین مسیر بین هر دو رأس از گراف را می‌توان با n بار اجرای الگوریتم دایسترا با استفاده از هیپ فیبوناچی، در زمان $O(n^2 \log n + nm)$ محاسبه کرد. [۵]

قضیّه‌ی ۴ . سطحِ P با n وجه داده شده است. با صرف هزینه‌ی زمانی $O(n \log n)$ می‌توان داده‌ساختاری ایجاد کرد که توسّط آن، به‌ازای هر نقطه‌ی دل‌خواه p روی سطح P، بتوان در زمان $O(\log n)$ وجهی را که p در آن قرار دارد، پیدا نمود. [۶]

۱.۴ الگوریتم تقسیم و حل

این الگوریتم برای پرس‌وجوی سریع‌تر، از یک مرحله‌ی پیش پردازش استفاده می‌کند که مراحل کلّی آن به صورت زیر است:

۱) با استفاده از الگوریتم Aleksandrov، گراف تقریبِ G را از روی چندوجهی P می‌سازیم. فرض می‌کنیم با اجرای این الگوریتم، بر روی هر یال گرافِ G به‌طور متوسّط m نقطه‌ی کمکی اضافه شده باشد.

۲) گراف G را به k ناحیه‌ی R_1، R_2، ...و R_k که هر ناحیه شامل $O(mn/k)$ رأس است، تقسیم می‌کنیم. گراف‌های وابسته به هر یک از این ناحیه‌ها را به‌ترتیب G_1، G_2، ...و G_k می‌نامیم.

۳) به ازای تمام ناحیه‌های R_i $(1 \leq i \leq k)$، کوتاه‌ترین مسیر بین هر دو رأس G_i را محاسبه می‌کنیم.

۴) برای هر ناحیه‌ی R_i، یک گراف کاملِ G'_i روی رأس‌های مرزی R_i می‌سازیم. وزن هر یال از G'_i نشان‌دهنده‌ی کوتاه‌ترین مسیر بین دو رأس مرزیِ G_i است، وقتی که این کوتاه‌ترین مسیر تماماً درون ناحیه‌ی R_i قرار داشته باشد.

۵) کوتاه‌ترین مسیر بین رأس‌های مرزیِ همه‌ی ناحیه‌ها، یعنی کوتاه‌ترین مسیر بین هر دو رأس از گراف $G'_1 \cup G'_2 \cup \cdots \cup G'_k$ را محاسبه می‌کنیم.

۶) فرض می‌کنیم $G''_i = G_i \cup G'_i$. بار دیگر برای هر ناحیه‌ی R_i، کوتاه‌ترین مسیر بین هر دو رأس از G''_i را محاسبه می‌کنیم.

فرض کنید می‌خواهیم کوتاه‌ترین مسیر بین دو رأس u و v از G را پیدا کنیم. دو حالت ممکن است رخ دهد: (الف) u و v هر دو در یک ناحیه، مثلاً R_i قرار دارند، (ب) u و v در دو ناحیه‌ی مختلف R_i و R_j قرار گرفته‌اند. در حالت اوّل کوتاه‌ترین مسیر بین u و v با دانستن کوتاه‌ترین مسیرها در G''_i به‌دست می‌آید. در حالت دوّم، کوتاه‌ترین مسیر بین u (و v) تا تمام رأس‌های مرزی R_i (و همچنین R_j) را در اختیار داریم. از طرفی کوتاه‌ترین مسیر بین تمام رئوس مرزی R_i و R_j را از قبل می‌دانستیم. ازآن‌جاکه $O(\sqrt{r})$ رأس مرزی در هر ناحیه وجود دارد ($r = O(mn/k)$)، می‌توانیم کوتاه‌ترین مسیر بین u و v را با مقایسه‌ی $O(\sqrt{r} \times \sqrt{r})$ مسیر به‌دست آوریم.

حال فرض کنید که می‌خواهیم کوتاه‌ترین مسیر بین دو نقطه‌ی s و t از P را بیابیم. اگر هر دو نقطه روی نقاط متناظر با رأس‌های G واقع شده باشند، کوتاه‌ترین مسیر را همانند بالا پیدا می‌کنیم. برای نقاط دل‌خواه دیگرِ P، مراحل زیر را انجام می‌دهیم:

۱) ناحیه‌های R_i و R_j را که شامل s و t هستند، پیدا می‌کنیم (ممکن است R_i و R_j برابر باشند).

۲) وجه‌های $f_i \in R_i$ و $f_j \in R_j$ را که به‌ترتیب شامل s و t هستند، می‌یابیم.

۳) از رأس s (t) یال‌های موقّتی به تمام نقاط کمکیِ f_i (f_j) در گراف G''_i (G''_j) اضافه می‌کنیم.

۴) کوتاه‌ترین مسیر از s (t) به تمام رأس‌های مرزیِ R_i (R_j) را محاسبه می‌کینم. با مقایسه‌ی مسیرهای ایجادشده توسّط زوج نقاط مرزیِ R_i و R_j، کوتاه‌ترین مسیر بین s و t را به‌دست می‌آوریم.

۲.۴ پیچیدگی زمانی الگوریتم

در مرحله‌ی ۱ از الگوریتم پیش پردازش، یک گرافِ G با حدّاکثر $O(nm)$ رأس و $O(nm^۲)$ یال می‌سازیم. این کار در زمان $O(nm^۲)$ قابل انجام است. در مرحله‌ی ۲، الگوریتم افراز با زمان خطی نسبت به تعداد رأس‌های G قابل انجام است، بنابراین زمان موردنیازِ این مرحله $O(nm)$ است. مرحله‌ی ۳، کوتاه‌ترین مسیر بین هر دو رأس را در هر ناحیه‌ی G_i محاسبه می‌کند. بنا بر قضیه‌ی ۳، این کار برای هر ناحیه هزینه‌ی $O(\frac{n^۲m^۲}{k^۲}\log(nm/k) + \frac{n^۲m^۳}{k^۲})$ و در مجموع هزینه‌ی $O(n^۲m^۲/k\log(nm/k) + n^۲m^۳/k)$ در بر خواهد داشت. مرحله‌ی ۴ در زمان $O(\sqrt{nm/k} \times \sqrt{nm/k}) = O(nm/k)$ قابل انجام است. در مرحله‌ی ۵ باید کوتاه‌ترین مسیر را بین هر دو رأس گرافی با $O(\sqrt{knm})$ رأس و $O(nm)$ یال پیدا کنیم که این کار به زمان $O(knm\log\sqrt{knm} + nm\sqrt{knm})$ نیاز دارد. مرحله‌ی ٦ نیز همانند مرحله‌ی ۳ است و به همان زمانِ اجرا نیاز دارد. در نتیجه پیچیدگی زمانی مرحله‌ی پیش پردازش عبارت است از: $O(n^۲m^۲/k\log(nm/k) + n^۲m^۳/k + knm\log\sqrt{knm} + nm\sqrt{knm})$.

در مرحله‌ی ۱ و ۲ از الگوریتم پرس‌وجو، وجه‌هایی که شامل نقاط مبدأ و مقصد هستند پیدا می‌کنیم. این کار طبق قضیه‌ی ۴ در زمان $O(\log n)$ قابل انجام است. در مرحله‌ی ۳ و ۴، کوتاه‌ترین مسیر از s و t به رأس‌های مرزی ناحیه‌های شامل آن نقاط با $O(m\sqrt{nm/k})$ مقایسه به دست می‌آید و سپس با مقایسه‌ی $O(nm/k)$ مسیر ایجادشده بین زوج نقاط مرزی دو ناحیه، کوتاه‌ترین مسیر بین s و t محاسبه می‌شود. در نتیجه پیچیدگی زمانی مرحله‌ی پرس و جو عبارت است از: $O(nm/k + m\sqrt{nm/k})$. همچنین کلّ حافظه‌ی موردنیاز برای این الگوریتم برابر $O(n^۲m^۲/k)$ است.

۵ الگوریتم موازی

در این بخش به توضیح نحوه‌ی پیاده‌سازی الگوریتم موازی خواهیم پرداخت. برای پیاده‌سازی این الگوریتم از یک محیط پردازش موازی استفاده شده است که در آن پردازنده‌ها از طریق ارسال پیام[۴] با یک‌دیگر در ارتباط هستند. برای پیاده‌سازی موازی الگوریتم فرض می‌کنیم که $k + ۱$ پردازنده در اختیار داریم که از ۱ تا $k + ۱$ شماره‌گذاری شده‌اند. پردازنده‌ی شماره‌ی $k + ۱$ را به‌عنوان پردازنده‌ی مرکزی درنظر می‌گیریم و پردازنده‌های شماره‌ی ۱ تا k با این پردازنده در ارتباط خواهند بود.

الگوریتم موازی شامل دو مرحله است: مرحله‌ی پیش پردازش که طی آن اطلاعات موردنیاز برای پرس‌وجوی کوتاه‌ترین مسیر بین هر دو نقطه از سطح چندوجهی تولید و نگه‌داری می‌شود و مرحله‌ی پرس‌وجو که طی آن کوتاه‌ترین مسیر بین نقاط داده‌شده محاسبه می‌گردد. مرحله‌ی پیش پردازش که مراحل کلّی آن بسیار شبیه به الگوریتم ارائه شده در بخش ۴ می‌باشد، به شرح زیر است:

۱) ابتدا پردازنده‌ی مرکزی اطلاعات مربوط به چندوجهیِ P را می‌خواند و به‌ازای هر نقطه در P، یک رأس در گراف تقریب G، و بین هر دو نقطه که در چندوجهی به یک‌دیگر متصل‌اند، یک یال در G قرار می‌دهد.

۲) پردازنده‌ی مرکزی با استفاده از الگوریتم افراز گراف، G را به k ناحیه‌ی $R_۱$، $R_۲$، ...و R_k تقسیم می‌کند. سپس اطلاعات مربوط به ناحیه‌ی R_i را که شامل وجه‌های قرار گرفته در آن ناحیه است، به پردازنده‌ی i اُم می‌فرستد.

۳) پردازنده‌ی i اُم $(۱ \le i \le k)$ با دریافت اطلاعات ناحیه‌ی R_i، گراف G_i را با افزودن نقاط کمکی و یال‌های بین نقاط کمکی در هر وجه می‌سازد.

۴) هر یک از پردازنده‌های ۱ تا k، کوتاه‌ترین مسیر بین هر دو رأس از G_i را محاسبه می‌کند.

۵) هر پردازنده کوتاه‌ترین مسیر به‌دست آمده بین رأس‌های مرزیِ خود را به پردازنده‌ی مرکزی می‌فرستد. در این مرحله رأس‌های کمکیِ اضافه شده روی یال‌های مرزی نیز به‌عنوان رأس‌های مرزی درنظر گرفته می‌شوند.

٦) پردازنده‌ی مرکزی با دریافت کوتاه‌ترین مسیر بین رأس‌های مرزی ناحیه‌ها، کوتاه‌ترین مسیر بین هر دو رأس از رأس‌های مرزیِ همه‌ی ناحیه‌ها را محاسبه می‌کند و کوتاه‌ترین مسیر جدید به‌دست آمده بین هر دو رأس از رأس‌های مرزی ناحیه‌ی R_i را به پردازنده‌ی i اُم می‌فرستد.

۷) پردازنده‌ی i اُم $(۱ \le i \le k)$ با دریافت فاصله‌ی جدید بین رأس‌ها مرزی خود، بار دیگر کوتاه‌ترین مسیر بین هر دو رأس از رأس‌های درون ناحیه‌ی خود را محاسبه می‌کند.

[۴] Message Passing

پس از انجام مرحله‌ی پیش پردازش، سیستم آماده است که کوتاه‌ترین مسیر بین هر دو نقطه‌ی دل‌خواه روی سطح چندوجهی را محاسبه نماید. برای یافتن کوتاه‌ترین مسیر بین دو نقطه‌ی s و t از P مراحل زیر انجام می‌گیرد:

۱) پردازنده‌ی مرکزی ناحیه‌های R_i و R_j را که شامل s و t هستند، پیدا می‌کند (ممکن است R_i و R_j برابر باشند). سپس نقاط s و t را به پردازنده‌های R_i و R_j ارسال می‌کند.

۲) پردازنده‌های i و j، وجه‌های f_i و f_j را که به‌ترتیب شامل s و t هستند، پیدا می‌کنند. سپس از رأس‌های s و t، یال‌های موقّتی به تمام نقاط کمکیِ f_i و f_j اضافه می‌شوند و کوتاه‌ترین مسیر از s و t به تمام رأس‌های مرزیِ R_i و R_j محاسبه و به پردازنده‌ی مرکزی ارسال می‌شود.

۳) پردازنده‌ی مرکزی با دریافت اطلاعات مربوط به فاصله‌ی نقاط s و t تا رأس‌های مرزیِ R_i و R_j و با داشتن کوتاه‌ترین مسیرها بین هر دو رأس از رأس‌های مرزیِ R_i و R_j کوتاه‌ترین مسیر بین s و t را به‌دست می‌آورد.

۱.۵ پیچیدگی زمانی و حافظه

در مرحله‌ی پیش پردازش، پردازنده‌ی مرکزی اطلاعات چندوجهی را از ورودی دریافت می‌کند و آن را به k ناحیه تقسیم می‌کند. همان‌طور که در قضیه‌ی ۲ دیدیم، این کار در $O(n)$ قابل انجام است. همچنین پردازنده‌ی مرکزی یک بار کوتاه‌ترین مسیر بین هر دو رأس از رأس‌های مرزیِ همه‌ی ناحیه‌ها را محاسبه می‌کند که هزینه‌ی زمانی این کار نیز برابر $O(knm\log\sqrt{knm}+nm\sqrt{knm})$ است (m تعداد نقاط کمکی اضافه‌شده روی هر یال است).

سایر پردازنده‌ها در مرحله‌ی پیش‌پردازش ابتدا زیرگراف مربوط به خود را می‌سازند که این‌کار در زمان $O(nm^2/k)$ قابل انجام است. سپس هر پردازنده، دو بار کوتاه‌ترین مسیر بین هر دو رأس خود را محاسبه می‌کند که این کار هزینه‌ی $O(\frac{n^2m^2}{k^2}\log(nm/k)+\frac{n^2m^3}{k^2})$ را دربر دارد. در نتیجه زمان اجرای مرحله‌ی پیش پردازش در پردازنده‌ی مرکزی برابر $O(knm\log\sqrt{knm}+nm\sqrt{knm})$ و در سایر پردازنده‌ها برابر $O(\frac{n^2m^2}{k^2}\log(nm/k)+\frac{n^2m^3}{k^2})$ است.

در مرحله‌ی پرس‌وجو، پردازنده‌ی مرکزی ناحیه‌های شامل نقاط مبدأ و مقصد را پیدا می‌کند و پس از دریافت اطلاعات کوتاه‌ترین مسیرها بین s و t تا رأس‌های مرزی ناحیه‌ها، با $O(nm/k)$ مقایسه کوتاه‌ترین مسیر را به دست می‌آورد. بنابراین زمان اجرای مرحله‌ی پرس و جو در پردازنده‌ی مرکزی برابر $O(nm/k)$ و در دو پردازنده‌ی شامل s و t برابر $O(m\sqrt{nm/k})$ است.

پردازنده‌ی مرکزی کوتاه‌ترین مسیر بین هر دو رأس از رأس‌های مرزیِ همه‌ی ناحیه‌ها را نگه‌داری می‌کند. با توجّه به این که تعداد رأس‌ها مرزی هر ناحیه $O(\sqrt{nm/k})$ و در نتیجه تعداد کلّ رأس‌های مرزی $O(\sqrt{knm})$ است، میزان حافظه‌ی موردنیاز برای پردازنده‌ی مرکزی $O(knm)$ است. همچنین سایر پردازنده‌ها باید کوتاه‌ترین مسیر بین تمام رأس‌های درون ناحیه‌ی خود را حفظ کنند. ازآن‌جاکه هر ناحیه $O(nm/k)$ رأس دارد، فضای حافظه‌ی موردنیاز برای سایر پردازنده‌ها $O(\frac{n^2m^2}{k^2})$ است.

۶ نتایج پیاده‌سازی

در این بخش به بررسی نتایج حاصل از پیاده‌سازی و آزمون الگوریتم موازی که در بخش قبلی معرّفی شد، می‌پردازیم. برای این کار ابتدا فرضیاتی که در مورد نحوه‌ی آزمون و بستر اجرای آن وجود دارند، ذکر می‌کنیم و در ادامه به بیان نتیجه‌ی حاصل از آزمون الگوریتم موازی و میزان تسریع آن نسبت به الگوریتم ترتیبی می‌پردازیم.

۱.۶ فرضیات آزمون

الگوریتم موازی برای یافتن کوتاه‌ترین مسیر وزن‌دار، با استفاده از توابع کتاب‌خانه‌ای LAM/MPI، که یک کتاب‌خانه‌ی استاندارد از توابع انتقال اطلاعات، مبتنی بر ارسال پیام است، طراحی و در محیط سیستم عامل Linux پیاده‌سازی شده است. تعداد پردازنده‌های موجود و مورد استفاده در آزمون، چهار پردازنده‌ی پنتیوم با سرعت ۲۳۳MH و حافظه‌ی ۶۴MB است که از طریق یک خط ارتباطی

تسریع	الگوریتم موازی	الگوریتم ترتیبی	ϵ
1.56	16.0 sec	25.0 sec	0.4
1.76	37.0 sec	65.0 sec	0.3
2.21	138.0 sec	305.0 sec	0.2

جدول ۲: زمان اجرای مرحله‌ی پیش پردازش روی یک ۱۰۰وجهی تصادفی.

با سرعت $100 Mb/s$ به یک‌دیگر متصل شده‌اند. همچنین از زبان برنامه‌نویسی C++ به‌علاوه‌ی توابع کتاب‌خانه‌ای LEDA[5] برای پیاده‌سازی الگوریتم‌های هندسی و گراف و همچنین طراحی داده‌ساختارهای موردنیاز استفاده شده است.

برای آزمون الگوریتم موازی کوتاه‌ترین مسیر، از چندوجهی‌های تصادفی استفاده شده است. به‌ازای هر چندوجهی، ۱۰۰ زوج نقطه‌ی تصادفی به‌عنوان نقاط مبدأ و مقصد مورد پرس‌وجو واقع و میانگین زمان اجرا به‌عنوان نتیجه‌ی نهایی ثبت شده است.

۲.۶ نتایج آزمون

همان‌طور که در بخش ۴ دیدیم، الگوریتم‌های پرس‌وجوی کوتاه‌ترین مسیر شامل دو مرحله‌ی اصلی هستند: مرحله‌ی پیش پردازش و مرحله‌ی پرس‌وجو. با استفاده از الگوریتم موازی و با بهره‌گیری از چند پردازنده می‌توان زمان اجرای هر یک از این مراحل را کاهش داد.

۱.۲.۶ مرحله‌ی پیش پردازش

جدول ۲ زمان اجرای مرحله‌ی پیش پردازش الگوریتم را بر روی یک ۱۰۰وجهی تصادفی، در دو حالت ترتیبی و موازی (با ۴ پردازنده) نشان می‌دهد. گراف تقریب این چندوجهی بر اساس الگوریتم Aleksandrov ساخته شده و به‌ازای مقادیر مختلف ϵ مورد آزمون قرار گرفته است. بدیهی است که با کاهش درجه‌ی تقریب ϵ، متوسط تعداد نقاط کمکی روی یال‌ها و در نتیجه زمان اجرای الگوریتم افزایش می‌یابد. همچنین در این جدول میزان تسریع[6] که بیان‌گر نسبت زمان اجرای الگوریتم ترتیبی به زمان اجرای الگوریتم موازی است، ذکر گردیده است.

همان طور که در جدول ۲ مشاهده می‌شود، میزان تسریع با افزایش تعداد نقاط افزایش می‌یابد و به حدّ نهاییِ مورد انتظار که در این جا به‌دلیل وجود ۴ پردازنده برابر ۳ است، نزدیک می‌شود. به‌طور کلّی با داشتن $k+1$ پردازنده، حدّاکثر تسریعی که می‌توان از مرحله‌ی پیش پردازش به دست آورد، برابر k است. البته با توجّه به هزینه‌ای که صرف تبادل اطلاعات می‌شود، به دست آوردن تسریع k عملاً ممکن نیست ولی همان‌طور که در جدول ۲ دیدیم، هرچه تعداد نقاط افزایش می‌یابد، هزینه‌ی تبادل اطلاعات در مقابل هزینه‌ی پردازش اطلاعات قابل صرف نظر و میزان تسریع به عدد k نزدیک‌تر می‌شود.

۲.۲.۶ مرحله‌ی پرس‌وجو

جدول ۳ زمان اجرای مرحله‌ی پرس‌وجو را برای یک ۱۰۰وجهی تصادفی در دو حالت ترتیبی و موازی (با ۴ پردازنده) و به‌ازای مقادیر مختلف ϵ نشان می‌دهد. در پیاده‌سازی مرحله‌ی پرس‌وجو، علاوه‌بر این که کوتاه‌ترین مسیر بین نقطه‌ی مبدأ s و نقطه‌ی مقصد t تا مرزهای ناحیه‌هایی که s و t در آن واقع شده‌اند، به‌طور موازی در پردازنده‌های مربوط به دو ناحیه محاسبه می‌شوند، برای رسیدن به تسریع بالاتر هم‌زمان با پردازش این اطلاعات توسط پردازنده‌ی مرکزی، کوتاه‌ترین مسیرهای مربوط به نقاط مبدأ و مقصد پرس‌وجوی بعدی، به پردازنده‌های مربوط به ناحیه‌های آن‌ها واگذار گردیده است. عملاً در هر لحظه، عملیات مربوط به دو پرس‌وجو به‌طور هم‌زمان در حال انجام است. در این آزمون با توجّه به محدودیت پردازنده‌های در دست‌رس (مجموعاً ۴ پردازنده) عملیات موازی‌سازی تا حد فوق مورد آزمایش قرار گرفته است. چنان‌چه پردازنده‌های بیش‌تری در اختیار باشند، می‌توان سطح موازات را به میزان بیش‌تری افزایش داد.

[5] Library of Efficient Datastructures and Algorithms

[6] Speed Up

ϵ	الگوریتم ترتیبی	الگوریتم موازی	تسریع
0.4	0.23 sec	0.20 sec	1.15
0.3	0.30 sec	0.23 sec	1.30
0.2	0.67 sec	0.40 sec	1.67

جدول ۳: زمان اجرای مرحله‌ی پرس‌وجو روی یک ۱۰۰وجهی تصادفی.

۷ نتیجه‌گیری

در این مقاله، مسئله‌ی پرس و جوی کوتاه‌ترین مسیر بین دو نقطه از سطح یک چندوجهی وزن‌دار را مورد بررسی قرار دادیم و یک الگوریتم موازی برای حلّ این مسئله ارائه نمودیم. سپس با بررسی نتایج عملی، میزان تسریع آن را نسبت به الگوریتم ترتیبی بررسی کردیم.

همان‌طور که ذکر شد، با در اختیار داشتن تعداد پردازنده‌های بیش‌تر، می‌توان سطح موازات الگوریتم را در مرحله‌ی پرس‌وجو افزایش داد. یکی از روش‌هایی که می‌توان برای دست‌یابی به این منظور پیش‌نهاد نمود این است که تعداد پردازنده‌های مرکزی را از یک، به دو یا چند پردازنده افزایش دهیم تا از تراکم کار روی پردازنده‌ی مرکزی و بیکار ماندن سایر پردازنده‌ها جلوگیری به عمل آوریم. همچنین می‌توانیم با ترکیب اعمال مربوط به چند پرس‌وجو، به‌طور هم‌زمان از پردازنده‌های بیکار برای انجام محاسبات مربوط به چند پرس‌وجو استفاده کنیم.

مراجع

[1] L. Aleksandrov, M. Lanthier, A. Maheshwari and J. R. Sack, *"An ϵ-Approximation Algorithm for Weighted Shortest Paths on Polyhedral Surfaces"*, SWAT 98, Stockholm, Sweden, July 1998.

[2] L. Aleksandrov, M. Lanthier, A. Maheshwari and J. R. Sack, *"An ϵ-Approximation Algorithm for Weighted Shortest Path Queries on Polyhedral Surfaces"*, 14th European Workshop on Computational Geometry, Barcelona, Spain, March 1998, pp. 19-21.

[3] L. Aleksandrov and H. Djidjev, *"Linear Algorithms for Partitioning Embeded Graphs of Bounded Genus"*, SIAM J., 9(1), 1996, pp. 129-150.

[4] E. Cohen, *"Efficient Parallel Shortest Paths in Digraphs with a Separator Decomposition"*, SPPA'93, 1993, pp. 57-67.

[5] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Press, 1990.

[6] M. De Berg, M. V. Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer, 1997

[7] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers, San Mateo, USA, 1992.

[8] R. C. Paige and C. P. Kruskal, *"Parallel Algorithms for Shortest Path Problems"*, In Proc. of 1989 International Conference on Parallel Processing, 1989, pp. 14-19.

[۹] حمید ضرّابی‌زاده، *«پیدا کردن موازی کوتاه‌ترین مسیر در سطوح نامنظّم مثلّث‌بندی‌شده‌ی وزن‌دار»*، پایان‌نامه‌ی کارشناسی ارشد، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران، آبان ماه ۱۳۷۹.

[۱۰] محمّد قدسی و حمید ضرّابی‌زاده، *«پیدا کردن کوتاه‌ترین مسیر در سطوح نامنظّم مثلث‌بندی‌شده‌ی وزن‌دار»*، مجموعه مقالات ششمین کنفرانس سالانه‌ی انجمن کامپیوتر ایران، اصفهان، اسفند ۱۳۷۹، صص ۲۱۷—۲۱۲.