به‌نام خدا

پوشش دیسک واحد

نام درس

نام و نام خانوادگی

دانشگاه صنعتی امیرکبیر، دانشکده مهندسی کامپیوتر

خردادماه ۱۴۰۱

چکیده

در نهایت عملکرد هر الگوریتم و بهترین الگوریتم‌ها برای هر معیار گزارش می‌شود.

۱ شرح مسئله

مجموعه P شامل n نقطه در صفحه داده می‌شود. هدف، پوشش تمام نقاط با استفاده از کمترین تعداد دیسک با شعاع واحد ($r = 1$) است. این مسئله، پوشش دیسک واحد (UDC)[۱] نامیده شده و یک مسئله ان‌پی‌سخت[۲] به‌حساب می‌آید [۱]. در کاربردهای مختلف مانند شبکه‌های بی‌سیم، تعیین موقعیت، برنامه‌ریزی حرکت، پردازش تصاویر و... استفاده می‌شود.

۲ مرور الگوریتم‌ها

از سال ۱۹۹۱ تاکنون الگوریتم‌های زیادی ارائه شده‌اند که این مسئله را به‌صورت تقریبی با فاکتورهای تقریب و پیچیدگی‌های زمانی متفاوت، در نُرم اقلیدسی حل می‌کنند. جدول ۱ تاریخچه الگوریتم‌های تقریبی ارائه شده برای این مسئله را نشان می‌دهد.

در ادامه، به‌ترتیب الگوریتم‌های ذیل بررسی می‌شوند:

- الگوریتم $BLMS$ که در سال ۲۰۱۷ ارائه شده است [۷].
- الگوریتم LL که در سال ۲۰۱۴ ارائه شده است [۶].
- الگوریتم DGT که در سال ۲۰۱۸ ارائه شده است [۹].
- الگوریتم $FastCover$ که در سال ۲۰۱۹ ارائه شده است [۱۰].

عنوان سه الگوریتم اول، برگرفته از حرف اول نام پژوهشگران مربوط به آن است.

[1] Unit Disk Cover

[2] NP-hard

جدول ۱: خلاصه الگوریتم‌های تقریبی ارائه شده برای UDC

مرجع	فاکتور تقریب	پیچیدگی زمانی	سال
[۲]	$2(1+\frac{1}{l})$	$O(l^2n^7)$	۱۹۹۱
[۲]	8	$O(n\log S)$	۱۹۹۱
[۳]	$O(1)$	$O(n^3\log n)$	۱۹۹۵
[۴]	$3(1+\frac{1}{l})^2$	$O(Kn)$	۲۰۰۱
[۵]	2.8334	$O(n(\log n\log\log n)^2)$	۲۰۰۷
[۶]	$25/6$	$O(n\log n)$	۲۰۱۴
[۷]	4	$O(n\log n)$	۲۰۱۷
[۸]	4	$O(n\log n)$	۲۰۱۸
[۹]	$O(1.321^d)$	$-$	۲۰۱۸
[۱۰]	7	$O(n)$	۲۰۱۹

۲-۱ الگوریتم $BLMS$

مجموعه P را به‌عنوان مجموعه نقاط ورودی که در صفحه قرار دارند و C^* را پوشش دیسک بهینه برای آن در نظر بگیرید. به‌خاطر داشته باشید که شعاع هر دیسک واحد برابر با ۱ است.

تعریف ۲-۱. در گراف تقاطع دیسک واحد[۳] $UDIG(P)$ نقاط موجود در مجموعه P رئوس را تشکیل می‌دهند و برای هر جفت $p,q\in P$ یک یال وجود دارد اگر و تنها اگر $|pq|\leqslant 2$ باشد که $|pq|$ فاصله اقلیدسی بین p و q را نشان می‌دهد.

مشاهده ۲-۲. برای دو نقطه $p,q\in P$ ، اگر $(p,q)\notin UDIG(P)$ آنگاه p و q نمی‌توانند با یک دیسک واحد پوشش داده شوند.

تعریف ۲-۳. یک مجموعه مستقل در $UDIG(P)$ ، زیرمجموعه‌ای مانند I از مجموعه P است، به‌طوری که هیچ یالی بین جفت نقطه‌های موجود در I وجود ندارد. همچنین I مجموعه مستقل حداکثری[۴] نامیده می‌شود، اگر برای هر $p\in P\setminus I$ ، مجموعه $I\cup\{p\}$ در $UDIG(P)$ مستقل نباشد.

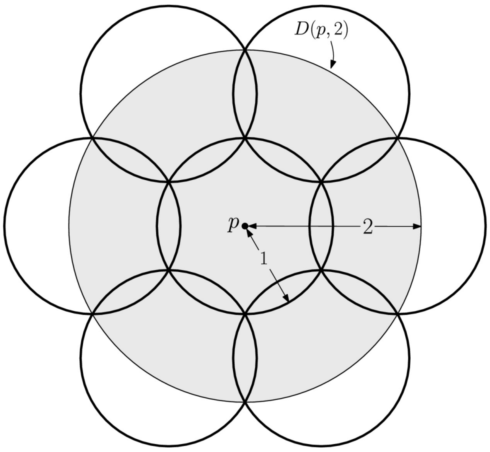
فرض کنید I مجموعه مستقل حداکثری در $UDIG(P)$ باشد. طبق مشاهده ۲-۲ اندازه هر مجموعه مستقل در $UDIG(P)$ یک حد پایین[۵] برای تعداد دیسک‌های موردنیاز به‌منظور پوشش P است. بنابراین خواهیم داشت:

$$|I|\leqslant|C^*| \tag{۱}$$

[3] Unit Disk Intersection Graph
[4] Maximal Independent Set
[5] Lower Bound

مطابق شکل ۱ برای پوشش یک دیسک با شعاع ۲ ، هفت دیسک واحد با شعاع ۱ لازم و کافی است. بر همین اساس، یک الگوریتم با فاکتور تقریب ۷ برای مسئله UDC به‌دست می‌آید.

فرض کنید I یک مجموعه مستقل حداکثری دلخواه در $UDIG(P)$ باشد. برای هر نقطه $p \in I$ فرض کنید $D(p,2)$ یک دیسک با مرکزیت نقطه p و شعاع ۲ باشد. همچنین درنظر داشته باشید که $d(p)$ دیسک واحدی است که نقطه p را پوشش می‌دهد. علاوه‌براین، تمام نقاطی که از مجموعه P توسط $d(p)$ پوشش داده می‌شوند، در $D(p,2)$ وجود دارند. بنابراین با پوشش $D(p,2)$ به‌وسیله ۷ دیسک واحد، به‌ازای همه $p \in I$ ، الگوریتمی با فاکتور تقریب ۷ حاصل می‌شود. باید توجه داشت که $UDIG(P)$ ممکن است حداکثر $O(n^2)$ یال داشته باشد. بنابراین پیچیدگی زمانی محاسبه $UDIG(P)$ در بدترین حالت درجه دو خواهد بود.



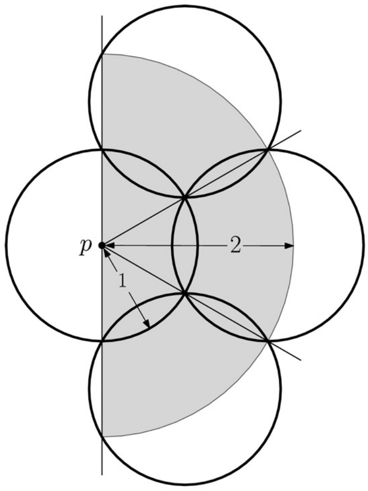
شکل ۱: پوشش $D(p,2)$ با دیسک‌های واحد

در ادامه خواهید دید که چگونه می‌توان فاکتور تقریب را به ۴ کاهش داد. p را سمت چپ‌ترین نقطه در مجموعه P درنظر بگیرید. در موارد خاص که مقادیر x نقاط با هم برابر است، برای انتخاب سمت چپ‌ترین نقطه، کمتر بودن مقدار y را ملاک قرار می‌دهیم. فرض کنید خط عمودی l از نقطه p عبور کند؛ $R(p)$ اشتراک $D(p,2)$ با نیم‌صفحه[6] سمت راست l خواهد بود؛ یعنی $R(p)$ نیم‌دیسک سمت راست $D(p,2)$ است (مطابق شکل ۲).

همان‌طور که قبلاً هم توضیح داده شد، تمام نقاط مجموعه P که با $d(p)$ پوشش داده شده‌اند، در $D(p,2)$ و به‌تبع آن در $R(p)$ نیز قرار دارند. مطابق شکل ۲، نیم‌دیسک $R(p)$ می‌تواند با ۴ دیسک واحد پوشش داده شود. قسمت دوم شکل، حالتی از موقعیت قرار گرفتن ۷ نقطه را نشان می‌دهد که برای پوشش آنها حداقل به ۴ دیسک واحد نیاز است.

برای نقطه‌ای مانند p و مجموعه نقاط I فاصله $d(p,I)$ برابر است با کمترین فاصله اقلیدسی بین p و هر

[6] Half-plane



شکل ۲: پوشش $R(p)$ با دیسک‌های واحد

نقطه موجود در I . اگر مجموعه I تهی باشد، فاصله بینهایت درنظر گرفته می‌شود. الگوریتم ۱، الگوریتم پیشنهادی با فاکتور تقریب ۴ را نشان می‌دهد. خروجی آن، مجموعه‌ای از دیسک‌های واحد با نام C است که مجموعه نقاط P را پوشش می‌دهند. ابتدا لیستی از نقاط که از چپ به راست مرتب شده‌اند ایجاد می‌شود. سپس هردفعه اولین عنصر p از لیست انتخاب و حذف می‌شود. اگر فاصله $d(p, I) \leqslant 2$ باشد، نشان‌دهنده این است که قبلاً نقطه p توسط یکی از دیسک‌های مجموعه C پوشش داده شده است. در غیر این صورت، نیم‌دیسک $R(p)$ با ۴ دیسک واحد پوشش داده می‌شود و به مجموعه C اضافه می‌شود. در نهایت مجموعه دیسک‌های واحد C به‌عنوان خروجی حاصل می‌شود.

الگوریتم ۱ نسخه اولیه $BLMS$

$C = \varnothing$
$I = \varnothing$
$L =$ List of points in P sorted from left to right
while L is not empty **do**
    $p =$ first element of L
    if $d(p, I) > 2$ **then**
        Cover $R(p)$ by 4 unit disks c_1, c_2, c_3, c_4
        $C = C \cup \{c_1, c_2, c_3, c_4\}$
        $I = I \cup \{p\}$
    $L = L - \{p\}$
return C

در هر تکرار الگوریتم ۱ ، نقطه p به مجموعه I اضافه می‌شود، اگر و تنها اگر $d(p, I) > 2$ باشد. بنابراین p در $UDIG(P)$ به هیچ نقطه‌ای از I متصل نیست. همچنین بعد از خاتمه الگوریتم، I یک مجموعه مستقل حداکثری خواهد بود.

قضیه ۲-۴. فاکتور تقریب الگوریتم ۱ برای مسئله پوشش دیسک واحد، ۴ است.

برهان. مجموعه نقاط I و مجموعه دیسک‌های واحد C را در پایان الگوریتم، درنظر بگیرید. براساس رابطه ۱ می‌دانیم که نامساوی $|I| \leqslant |C^*|$ برقرار است. به‌ازای نقاط $p \in I$ هر نقطه $q \in P$ در یک نیم‌دیسک $R(p)$ قرار دارد (ممکن است $q = p$ باشد). چون برای هر نقطه $p \in I$ ، نیم‌دیسک $R(p)$ با ۴ دیسک واحد پوشش داده می‌شود، مجموعه C مجموعه P را پوشش می‌دهد. بنابراین رابطه $|C| \leqslant 4|I| \leqslant 4|C^*|$ برقرار است.

پیچیدگی زمانی الگوریتم ۱ $O(n\log n + n.t(d))$ است. $t(d)$ پیچیدگی زمانی محاسبه فاصله $d(p, I)$ را نشان می‌دهد. محاسبه این فاصله با پیچیدگی زمانی $O(\log^2 n)$ قابل انجام است [۱۱]. بنابراین پیچیدگی نهایی الگوریتم، $O(\log^2 n)$ است که در ادامه با استفاده از تکنیک جاروی صفحه، بهبود می‌یابد.

به‌جای محاسبه فاصله $d(p, I)$ کافی است بدانیم فاصله نقطه p از مجموعه I از ۲ بیشتر است یا نه؛ یعنی به یک مسئله تصمیم‌گیری تبدیل شود. به‌تدریج که یک نقطه جدید مانند p به مجموعه I اضافه می‌شود، نقاط مجموعه P که در نیم‌دیسک $R(p)$ قرار دارند، حذف می‌شوند. برهمین اساس، یک الگوریتم با استفاده از تکنیک جاروی صفحه و پیچیدگی زمانی $O(n\log n)$ ارائه می‌شود.

در الگوریتم ۲، ابتدا نقاط بر اساس مؤلفه x از چپ به راست مرتب شده و در صف رخدادها درج می‌شوند. خط جارو به‌صورت عمودی از چپ به راست بر روی نقاط حرکت می‌کند. به هر نقطه که می‌رسد، اگر نقطه انتهایی یک نیم‌دیسک باشد، نقطه ابتدایی مربوط به آن نیم‌دیسک را از درخت وضعیت حذف می‌کند. در غیر این صورت، دو نقطه همسایه از بالا و دو نقطه همسایه از پایین را در درخت وضعیت برای این نقطه پیدا می‌کند. اگر فاصله آن با حداقل یکی از این ۴ همسایه کمتر یا مساوی ۲ باشد، به این معنا است که نقطه مورد نظر قبلاً در یک نیم‌دیسک قرار گرفته و تحت پوشش است و نیاز به اقدام خاصی نیست. در غیر این صورت، یک نیم‌دیسک به شعاع ۲ و مرکز آن نقطه درنظر گرفته می‌شود و مرکز ۴ دیسک واحد پوشش‌دهنده آن نیم‌دیسک محاسبه و به‌عنوان جواب گزارش می‌شود. سپس آن نقطه به‌عنوان نقطه ابتدایی نیم‌دیسک، در مکان مناسب خود در درخت وضعیت درج می‌شود. همچنین، نقطه انتهایی این نیم‌دیسک با اضافه کردن ۲ واحد به مؤلفه x مرکز آن، محاسبه شده و در جای مناسب در صف رخدادها درج می‌شود. لازم به ذکر است که درخت وضعیت، شامل نقاط ابتدایی نیم‌دیسک‌هایی است که در هر لحظه با خط جارو متقاطع‌اند و به‌صورت مرتب شده بر اساس مؤلفه y از پایین به بالا قرار گرفته‌اند.

الگوریتم ۲ نسخه بهبود یافته $BLMS$ با تکنیک جاروی صفحه

Initialize an empty event queue Q. Insert the points in ascending order of their x-coordinates into Q.
Initialize an empty BST status structure T.
Initialize an empty list C.
while Q is not empty **do**
    Determine the next event point p in Q and delete it.
    if p is an end-point **then**
        Delete the start-point of the corresponding half-disk from T.
    else
        Find the 2 top and the 2 bottom neighbors of p in T.
        if The distance between p and all of these 4 neighbors is greater than 2 **then**
            Calculate center points of the 4 unit disks which cover half-disk of point p and insert them into C.
            Insert p into T.
            Insert the end-point $q = (p_x + 2, p_y)$ into Q.
return C

۲-۲ الگوریتم LL

در این الگوریتم، صفحه به نوارهای عمودی با عرض $\sqrt{3}$ تقسیم می‌شود. از هر نوار، یک جواب تقریبی با مرتب‌سازی نقاط براساس مؤلفه y به‌صورت نزولی به‌دست می‌آید. نقطه بعدی درون یک نوار که هنوز پوشش داده نشده است، با قرار دادن یک دیسک در پایین‌ترین مکان ممکن، پوشش داده می‌شود. مرکز این دیسک‌ها، روی خطوط عمودی که نوارها را به دو قسمت تقسیم می‌کنند قرار می‌گیرد. جواب نهایی با اجتماع جواب همه نوارها حاصل می‌شود. این سامانه نواری، ۵ مرتبه به سمت راست و هر دفعه به‌اندازه $\sqrt{3}/6$ شیفت داده می‌شود. در هر شیف، یک جواب به‌دست می‌آید. از بین این ۶ جواب، آن جوابی که کمترین دیسک را استفاده کرده باشد به‌عنوان جواب نهایی درنظر گرفته می‌شود. جزئیات بیشتر در الگوریتم ۳ موجود است. فاکتور تقریب این الگوریتم $25/6 \approx 4.17$ است.

الگوریتم ۳ محاسبه موقعیت دیسک‌های واحد با استفاده از LL

Disk-Centers $\leftarrow \emptyset$, $\texttt{min} \leftarrow n+1$;
Sort P w.r.t x-coordinate in $O(n \log n)$ time;
for $i \in \{0,1,2,3,4,5\}$ **do**
  $\texttt{current} \leftarrow 1$, $C \leftarrow \emptyset$, $\texttt{right} \leftarrow P[1]_x + \frac{i\sqrt{3}}{6}$;
  while $\texttt{current} \leq n$ **do**
    $\texttt{index} \leftarrow \texttt{current}$;
    while $P[\texttt{current}]_x <$ right **and** $\texttt{current} \leq n$ **do**
      $\texttt{current} \leftarrow \texttt{current} + 1$;
    x-$\texttt{of-restriction-line} \leftarrow \texttt{right} - \sqrt{3}/2$, $\texttt{segments} \leftarrow \emptyset$;
    for $j \leftarrow \texttt{index}$ **to** $\texttt{current}-1$ **do**
      $d \leftarrow P[j]_x - x$-$\texttt{of-restriction-line}$, $y \leftarrow \sqrt{1-d^2}$;
      Create a segment s having the endpoints (x-$\texttt{of-restriction-line}$, $P[j]_y + y$) and (x-$\texttt{of-restriction-line}$, $P[j]_y - y$) and insert it into $\texttt{segments}$;
    Sort $\texttt{segments}$ in non-ascending order based on y-coordinates of their tops. Greedily stab them by choosing the stabbing point as low as possible, while still stabbing the topmost unstabbed segment. Put the stabbing points (the disk centers) in C;
    Increment $\texttt{right}$ by a multiple of $\sqrt{3}$ such that $P[\texttt{current}] - \texttt{right} \leq \sqrt{3}$;
  if $|C| < \texttt{min}$ **then**
    Disk-Centers $\leftarrow C$, $\texttt{min} \leftarrow |C|$;
return Disk-Centers;

۳-۲ الگوریتم DGT

این الگوریتم، ساده و برخط است. به‌ازای هرنقطه‌ای که تاکنون تحت پوشش قرار نگرفته است، یک دیسک واحد به مرکز آن نقطه ایجاد می‌شود. فاکتور تقریب آن در صفحه، ۵ است. در فضای با ابعاد d دارای فاکتور تقریب $O(1.321^d)$ است. برای مشاهده توصیف سطح بالا، به الگوریتم ۴ مراجعه نمایید.

الگوریتم ۴ محاسبه موقعیت دیسک‌های واحد با استفاده از DGT

Disk-Centers $\leftarrow \emptyset$;
for $p \in P$ **do**
  if the distance from p to the nearest point in Disk-Centers is > 1 **then**
    Disk-Centers $\leftarrow$ Disk-Centers $\cup \{p\}$;
return Disk-Centers;

۲-۴ الگوریتم $FastCover$

در این الگوریتم، از یک شبکه[۷] با مربع‌های به ضلع $\sqrt{2}$ استفاده می‌شود. هر مربع از این شبکه، می‌تواند توسط یک دیسک به شعاع واحد محاط شود. به‌ازای هر نقطه، اگر توسط یکی از دیسک‌هایی که قبلاً قرار گرفته است تحت پوشش باشد، عملی انجام نمی‌شود؛ در غیر این صورت، یک دیسک واحد به مرکز مربعی که آن نقطه درونش قرار گرفته است، ایجاد می‌شود.

در پیاده‌سازی برای جستجوی سریع‌تر، از یک جدول درهم‌ساز[۸] استفاده می‌شود. در این جدول، مختصات مرکز دیسک‌هایی که اضافه شده‌اند ذخیره می‌شود. برای جلوگیری از مشکلات اعداد اعشاری، از یک جفت عدد صحیح برای نمایش مرکز هر دیسک استفاده می‌شود. مختصات حقیقی می‌تواند با ضرب کردن هر عدد صحیح در $\sqrt{2}$ و اضافه کردن $\sqrt{2}/2 = 1/\sqrt{2}$ به آن به‌دست بیاید. برای به‌دست آوردن اعداد صحیح از روی یک نقطه، مؤلفه‌های x, y آن بر $\sqrt{2}$ تقسیم می‌شود. درواقع این اعداد صحیح، مربع مربوط به آن نقطه را در شبکه نشان می‌دهد. این فرآیند در الگوریتم ۵ قابل ملاحظه است.

این الگوریتم دارای فاکتور تقریب ۷ و پیچیدگی زمانی $O(n)$ است. یک الگوریتم برخط به‌حساب می‌آید و هیچ پیش‌پردازشی مثل مرتب‌سازی روی نقاط انجام نمی‌دهد. به‌اندازه $O(s)$ حافظه اضافی مصرف می‌کند که s نشان‌دهنده اندازه پوشش تولید شده است.

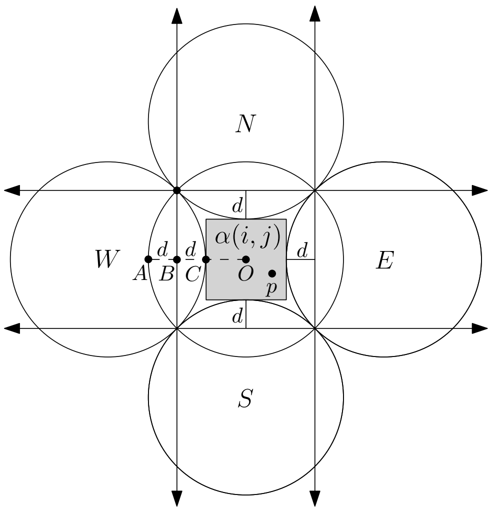
الگوریتم ۵ محاسبه موقعیت دیسک‌های واحد با استفاده از $FastCover$

1: $\mathcal{H} \leftarrow \emptyset$; Disk-Centers $\leftarrow \emptyset$;
2: **for** $p \in P$ **do**
3: $\quad i \leftarrow \lfloor p_x/\sqrt{2} \rfloor$; $j \leftarrow \lfloor p_y/\sqrt{2} \rfloor$;
4: $\quad$ **if** $(i, j) \notin \mathcal{H}$ **then**
5: $\quad\quad$ insert (i, j) into $\mathcal{H}$ and $(\sqrt{2}i + \frac{1}{\sqrt{2}}, \sqrt{2}j + \frac{1}{\sqrt{2}})$ into Disk-Centers;
6: **return** Disk-Centers;

این الگوریتم را می‌توان کمی بهبود داد. وقتی که یک نقطه در یک مربع از شبکه قرار می‌گیرد، ممکن است توسط ۴ دیسک واحد که مربوط به مربع‌های همسایه است و قبلاً اضافه شده‌اند تحت پوشش باشد (مطابق شکل ۳). بنابراین بهتر است برای کاهش تعداد دیسک‌ها این شرایط نیز بررسی شود. در الگوریتم ۶ جزئیات نسخه بهبودیافته ذکر شده است.

---

[7] Grid
[8] Hash table



شکل ۳: امکان پوشش یک نقطه با دیسک‌های مربع‌های همسایه

الگوریتم ۶ محاسبه موقعیت دیسک‌های واحد با استفاده از $FastCover+$

$\mathcal{H} \leftarrow \emptyset$; Disk-Centers $\leftarrow \emptyset$;
for $p \in P$ **do**
  $i \leftarrow \lfloor p_x/\sqrt{2} \rfloor$; $j \leftarrow \lfloor p_y/\sqrt{2} \rfloor$;
  if $(i,j) \in \mathcal{H}$ **then**
    update $B(i,j)$ using p; {p is already covered by $D(i,j)$}
  else if $p_x \geq \sqrt{2}(i+1.5)-1$ **and** $(i+1,j) \in \mathcal{H}$ **and**
    `distance`$(p, (\sqrt{2}(i+1)+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})) \leq 1$ **then**
    continue; {p is covered by the grid-disk E placed before}
  else if $p_x \leq \sqrt{2}(i-0.5)+1$ **and** $(i-1,j) \in \mathcal{H}$ **and**
    `distance`$(p, (\sqrt{2}(i-1)+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})) \leq 1$ **then**
    continue; {p is covered by the grid-disk W placed before}
  else if $p_y \geq \sqrt{2}(j+1.5)-1$ **and** $(i,j+1) \in \mathcal{H}$ **and**
    `distance`$(p, (\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}(j+1)+\frac{1}{\sqrt{2}})) \leq 1$ **then**
    continue; {p is covered by the grid-disk N placed before}
  else if $p_y \leq \sqrt{2}(j-0.5)+1$ **and** $(i,j-1) \in \mathcal{H}$ **and**
    `distance`$(p, (\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}(j-1)+\frac{1}{\sqrt{2}})) \leq 1$ **then**
    continue; {p is covered by the grid-disk S placed before}
  else
    insert (i,j) into $\mathcal{H}$ and $(\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})$ into Disk-Centers;
return Disk-Centers;

بهبود دیگری می‌توان روی الگوریتم اعمال نمود. اگر نقاط موجود در دو دیسک مجاور را بتوان با یک دیسک پوشش داد، یعنی قطر مجموعه نقاط هر دو دیسک کمتر از ۲ باشد، می‌توان آن دو دیسک را ادغام نمود. برای ادغام، یک دیسک واحد به مرکز وسط قطر مجموعه نقاط درنظر گرفته می‌شود. برای اینکه محاسبه قطر، تأثیری روی پیچیدگی زمانی الگوریتم نداشته باشد، به‌همراه هر دیسک واحدی که ایجاد

می‌شود، یک مستطیل مرزی نیز برای مجموعه نقاط موجود در آن نگهداری می‌شود. هر دفعه که یک نقطه جدید تحت پوشش یک دیسک واحد قرار می‌گیرد، ابعاد مستطیل مرزی مربوط به آن نیز بروزرسانی می‌شود. این بروزرسانی در $O(1)$ قابل انجام است. جزئیات بیشتر در الگوریتم ۷ ذکر شده است.

الگوریتم ۷ محاسبه موقعیت دیسک‌های واحد با استفاده از $FastCover++$

$\mathcal{H} \leftarrow \emptyset$; Disk-Centers $\leftarrow \emptyset$;
for $p \in P$ **do**
    $i \leftarrow \lfloor p_x/\sqrt{2} \rfloor$; $j \leftarrow \lfloor p_y/\sqrt{2} \rfloor$;
    if $(i,j) \in \mathcal{H}$ **then**
        update $B(i,j)$ using p; {p is already covered by $D(i,j)$}
    else if $p_x \geq \sqrt{2}(i+1.5)-1$ **and** $(i+1,j) \in \mathcal{H}$ **and**
        `distance`$(p, (\sqrt{2}(i+1)+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})) \leq 1$ **then**
        update $B(i+1,j)$ using p; {p is covered by the grid-disk E placed before}
    else if $p_x \leq \sqrt{2}(i-0.5)+1$ **and** $(i-1,j) \in \mathcal{H}$ **and**
        `distance`$(p, (\sqrt{2}(i-1)+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})) \leq 1$ **then**
        update $B(i-1,j)$ using p; {p is covered by the grid-disk W placed before}
    else if $p_y \geq \sqrt{2}(j+1.5)-1$ **and** $(i,j+1) \in \mathcal{H}$ **and**
        `distance`$(p, (\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}(j+1)+\frac{1}{\sqrt{2}})) \leq 1$ **then**
        update $B(i,j+1)$ using p; {p is covered by the grid-disk N placed before}
    else if $p_y \leq \sqrt{2}(j-0.5)+1$ **and** $(i,j-1) \in \mathcal{H}$ **and**
        `distance`$(p, (\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}(j-1)+\frac{1}{\sqrt{2}})) \leq 1$ **then**
        update $B(i,j-1)$ using p; {p is covered by the grid-disk S placed before}
    else
        insert (i,j) into $\mathcal{H}$ and initialize $B(i,j)$ using p;
while there is a grid-disk $(i,j) \in \mathcal{H}$ that is not considered yet **do**
    if there is a grid disk $(k,\ell) \in \mathcal{H}$ such that $|i-k| \leq 1, |j-\ell| \leq 1$ and the diagonal-length of the bounding-box $B := B(i,j) \cup B(k,\ell)$ is at most 2 **then**
        remove (i,j) and (k,ℓ) from $\mathcal{H}$ and add the center of B to Disk-Centers;
for every grid-disk $(i,j) \in \mathcal{H}$ **do**
    insert $(\sqrt{2}i+\frac{1}{\sqrt{2}}, \sqrt{2}j+\frac{1}{\sqrt{2}})$ into Disk-Centers;
return Disk-Centers;

---

۳ ارزیابی

همه الگوریتم‌ها با زبان $C++$ و کتابخانه $CGAL$ پیاده‌سازی شده‌اند. هرکدام از آنها بر روی ۱۰ مجموعه‌نقطه دنیای واقعی اجرا شده‌اند. جدول ۲ نتایج ارزیابی را نشان می‌دهد. هر الگوریتم، ۵ بار بر روی هر مجموعه‌نقطه اجرا شده است. هر خانه کمترین تعداد دیسک و کمترین زمان پردازش برحسب ثانیه از بین این ۵ اجرا را نشان می‌دهد. منظور از $LL-1P$ ، اجرای یک مرحله‌ای الگوریتم ۳ به‌جای شش مرحله است.

جدول ۲: نتایج ارزیابی الگوریتم‌ها

	LL	LL-1P	BLMS	DGT	FastCover	FastCover+	FastCover++
birch3	99989, 0.18	99991, 0.03	99994, 0.05	99993, 0.07	99996, 0.02	99995, 0.02	99980, 0.08
monalisa	100000, 0.15	100000, 0.03	100000, 0.11	100000, 0.08	100000, 0.02	100000, 0.02	100000, 0.08
usa	115475, 0.17	115475, 0.04	115475, 0.12	115475, 0.09	115475, 0.02	115475, 0.03	115475, 0.10
KDDCU2D	1147, 0.19	1152, 0.04	1692, 0.10	1626, 0.01	1418, 0.01	1374, 0.01	1257, 0.01
europe	168253, 0.35	168271, 0.06	168088, 0.19	168069, 0.16	168333, 0.03	168277, 0.04	167811, 0.20
wildfires	622, 3.74	622, 0.88	842, 1.21	787, 0.12	663, 0.04	637, 0.04	620, 0.06
world	6667, 2.84	6680, 0.51	9145, 0.79	10980, 0.15	7874, 0.03	7576, 0.04	6967, 0.07
nyctaxi	25, 13.84	26, 3.09	32, 2.90	31, 0.13	34, 0.05	31, 0.05	25, 0.10
uber	3, 21.06	3, 4.75	5, 4.03	5, 0.19	5, 0.06	4, 0.06	4, 0.16
hail2015	888, 39.83	889, 9.82	1193, 11.19	1128, 0.74	901, 0.28	860, 0.28	847, 0.42

۴ نتیجه‌گیری

در مجموعه‌نقطه‌های دنیای واقعی، اگر معیار با اهمیت‌تر برای ارزیابی، تعداد دیسک‌های استفاده شده باشد، الگوریتم LL عملکرد بهتری دارد. هرچند که الگوریتم $FastCover^{++}$ در برخی موارد، هم از لحاظ تعداد دیسک‌ها و هم زمان اجرا، عملکرد بهتری داشته است.

اگر معیار مهم‌تر، زمان اجرا باشد الگوریتم $FastCover$ عملکرد بهتری داشته است؛ اما چون تفاوت چندانی با نسخه بهبودیافته خود که تعداد دیسک‌های کمتری تولید می‌کند ندارد، استفاده از $FastCover^{++}$ پیشنهاد می‌شود.

اگر هردو معیار تعداد دیسک‌ها و زمان اجرا با اهمیت باشد، استفاده از الگوریتم $FastCover^{++}$ توصیه می‌شود؛ چرا که تعادل خوبی بین هردو معیار برقرار می‌کند [۱۲].

مراجع

[1] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, “Optimal packing and covering in the plane are np-complete,” *Information processing letters*, vol.12, no.3, pp.133–137, 1981.

[2] T. F. Gonzalez, “Covering a set of points in multidimensional space,” *Information processing letters*, vol.40, no.4, pp.181–188, 1991.

[3] H. Brönnimann and M. T. Goodrich, “Almost optimal set covers in finite vc-dimension,” *Discrete & Computational Geometry*, vol.14, no.4, pp.463–479, 1995.

[4] M. Franceschetti, M. Cook, and J. Bruck, “A geometric theorem for approximate disk covering algorithms,” 2001.

[5] B. Fu, Z. Chen, and M. Abdelguerfi, “An almost linear time 2.8334-approximation algorithm for the disc covering problem,” in *International Conference on Algorithmic Applications in Management*, pp.317–326, Springer, 2007.

[6] P. Liu and D. Lu, “A fast 25/6-approximation for the minimum unit disk cover problem,” *arXiv*, 2014.

[7] A. Biniaz, P. Liu, A. Maheshwari, and M. Smid, “Approximation algorithms for the unit disk cover problem in 2d and 3d,” *Computational Geometry*, vol.60, pp.8–18, 2017.

[8] M. Imanparast and S. N. Hashemi, “A simple greedy approximation algorithm for the unit disk cover problem,” *AUT Journal of Mathematics and Computing*, vol.1, no.1, pp.47–55, 2020.

[9] A. Dumitrescu, A. Ghosh, and C. D. Tóth, “Online unit covering in euclidean space,” *Theoretical Computer Science*, vol.809, pp.218–230, 2020.

[10] A. Ghosh, B. Hicks, and R. Shevchenko, “Unit disk cover for massive point sets,” in *International Symposium on Experimental Algorithms*, pp.142–157, Springer, 2019.

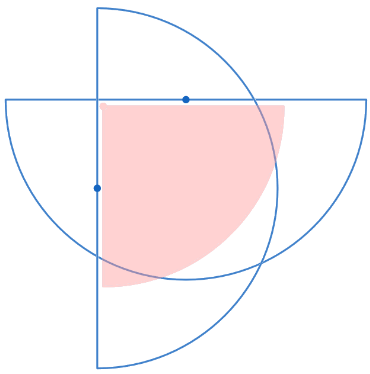
[11] J. L. Bentley and J. B. Saxe, “Decomposable searching problems i. static-to-dynamic transformation,” *Journal of Algorithms*, vol.1, no.4, pp.301–358, 1980.

[12] R. Friederich, M. Graham, A. Ghosh, B. Hicks, and R. Shevchenko, “Experiments with unit disk cover algorithms for covering massive pointsets,” *arXiv*, 2022.

پیوست

همان‌طور که در جدول ۱ ملاحظه شد، بهترین الگوریتم‌هایی که تاکنون برای مسئله UDC ارائه شده‌اند، دارای فاکتور تقریب ۴ هستند. اینجانب، توجه زیادی به این مسئله با هدف بهبود فاکتور تقریب و نگارش مقاله نمودم. ایده‌های مختلفی برای بهبود فاکتور تقریب به ذهنم رسید که پس از بررسی‌های فراوان متوجه شدم برخی از آنها اشتباه است و برخی دیگر را به‌دلیل کمبود زمان نتوانستم به‌طور دقیق بررسی کنم. در ادامه تعدادی از آنها را بیان می‌کنم.

ایده اول، با هدف کاهش فاکتور تقریب از ۴ به ۳ بود. در شکل ۲ نشان داده شد که برای پوشش یک نیم‌دیسک به شعاع ۲، دقیقاً به ۴ دیسک واحد احتیاج است. بر اساس این شکل، برای پوشش یک ربع‌دیسک نیز دقیقاً به ۳ دیسک واحد احتیاج است. با فرض اینکه اشتراک دو نیم‌دیسک به شعاع ۲ حداکثر با یک ربع‌دیسک قابل پوشش است، می‌توان یک‌بار مطابق الگوریتم ۲ خط جارو را از چپ به راست و بار دیگر از بالا به پایین حرکت داد. طی این دو مرحله، تعدادی نیم‌دیسک حاصل می‌شود. نواحی از نیم‌دیسک‌ها که با یکدیگر هم‌پوشانی پیدا می‌کنند (اشتراک نیم‌دیسک‌ها) مواردی است که لازم است با دیسک‌های واحد پوشش داده شود. پس از بررسی مشخص شد که این ایده عملی نیست. چون فرض اولیه اشتباه است و حالت‌هایی وجود دارد که اشتراک دو نیم‌دیسک بیشتر از حد تصور می‌شود و نمی‌توان آن را با یک ربع‌دیسک پوشش داد (مطابق شکل ۴).



شکل ۴: عدم امکان پوشش اشتراک دو نیم‌دیسک با ربع‌دیسک

ایده دوم نیز با هدف کاهش فاکتور تقریب از ۴ به ۳ بود. به‌جای نیم‌دیسک‌های با شعاع ۲، ربع‌دیسک‌هایی به شعاع ۲ درنظر گرفته می‌شود که هرکدام با ۳ دیسک واحد قابل پوشش هستند. در الگوریتم ۸ جزئیات ایده بیان شده است. در این الگوریتم، خط جارو از بالا به پایین بر روی نقاط حرکت می‌کند. پس از بررسی، مشخص شد که این ایده نیز، عملی نیست. زیرا مانند شکل ۵ حالت‌هایی وجود دارد که ربع‌دیسک‌های زیادی با یکدیگر همپوشانی پیدا می‌کنند و در نهایت فاکتور تقریب، $O(n)$ می‌شود.

الگوریتم ۸ ایده دوم برای کاهش فاکتور تقریب به ۳

Initialize an empty event queue Q. Insert the points in descending order of their y-coordinates into Q. If two points have the same y-coordinate, the one with smaller x-coordinate has higher priority.

Initialize an empty BST status structure T.

Initialize an empty list C.

while Q is not empty **do**

  Determine the next event point p in Q and delete it.

  if p is an end-point **then**

    Delete the start-point of the corresponding quarter disk from T.

  else

    Find the two left neighbors p', p'' of p in T. If p has the same x-coordinate with a point in T, consider the point with the higher y-coordinate as the left.

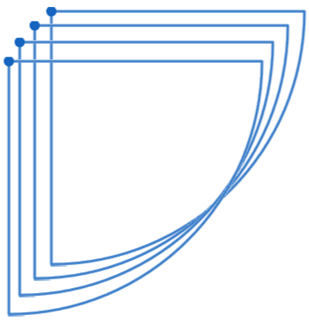
    if $|pp'| > 2$ and $|pp''| > 2$ **then**

      Calculate center points of the 3 unit disks which cover quarter disk of point p and insert them into C.

      Insert p into T.

      Insert the end-point $q = (p_x, p_y + 2)$ into Q.

return C



شکل ۵: حالتی که ربع‌دیسک‌های زیادی همپوشانی پیدا می‌کنند