# موازی سازی جستجوی نزدیک‌ترین همسایه در فضاهای با ابعاد بالا


**مجید قادری** **محمد قدسی**

دانشکدهٔ مهندسی کامپیوتر
دانشگاه صنعتی شریف
{majid@ce, ghodsi@}sharif.edu



**چکیده**

مجموعهٔ $P$ شامل $n$ نقطه در فضای حقیقی $\mathbb{R}^d$ داده شده است. فاصله بر اساس یک معیار Minkowski تعریف می‌شود. در این مقاله با استفاده از $O(n)$ پردازنده در مدل CREW PRAM یک BBD-tree متناظر با این نقاط در زمان $O(\log^2(n))$ ساخته می‌شود. تعداد گره‌های درخت از مرتبهٔ $O(n)$ و پیچیدگی هر گره $O(d)$ است. سپس بر اساس این درخت یک الگوریتم موازی برای جستجوی نزدیک‌ترین همسایه‌های نقاط $P$ ارائه می‌شود. زمان اجرای این الگوریتم $O(\log(n))$ خواهد بود.

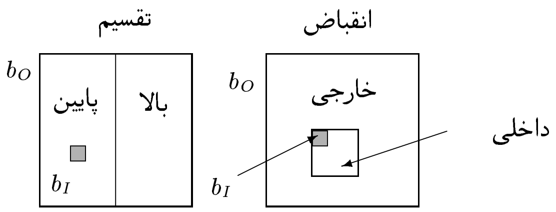

## ۱ مقدمه

مجموعهٔ $P$ شامل $n$ نقطهٔ $\{P_1, ..., P_n\}$ در فضای حقیقی $\mathbb{R}^d$ داده شده است. مسألهٔ نزدیک‌ترین همسایه‌ها (All Nearest Neighbors)، پیدا کردن نزدیک‌ترین همسایهٔ هر نقطه در این مجموعه است. نزدیکی بر اساس مفهوم فاصله تعریف می‌شود. فاصله می‌تواند هر معیار Minkowski باشد. به ازای هر عدد صحیح $m \geq 1$، $L_m$—فاصلهٔ بین دو نقطهٔ $p = (p_1, ..., p_d)$ و $q = (q_1, ..., q_d)$ به این صورت تعریف می‌شود

$$d(p, q) = \sqrt[m]{\sum_{i=1}^{d} (|p_{(i)} - q_{(i)}|)^m}$$

با این تعریف $L_2$ همان فضای اقلیدسی است.

مسألهٔ نزدیک‌ترین همسایه‌ها در زمینه‌های مختلف محاسباتی، به عنوان یک تکنیک اصلی به کار گرفته می‌شود. به عنوان مثال در مسائل بازیابی اطلاعات[۵]، تشخیص الگو، فشرده‌سازی داده‌ها[۶] و پایگاه‌های داده‌ای چند رسانه‌ای[۸] از این الگوریتم استفاده می‌شود. داده‌ها به صورت نقاطی در یک فضای چند بعدی نمایش داده می‌شوند. در این کاربردها معمولاً تعداد نقاط بسیار زیاد است خصوصاً اینکه ابعاد فضای جستجو نیز بالا است. به عنوان مثال کاربردهای بازیابی اطلاعات چند رسانه‌ای، شامل هزاران نقطه در صدها بعد[۸] است.




شکل ۱: جعبهٔ داخلی و خارجی

پیدا کردن نزدیک‌ترین همسایه در چنین کاربردهایی عمدتاً مستلزم صرف زمان زیادی است. زیرا هم تعداد نقاط زیاد است و هم تعداد ابعاد فضای جستجو زیاد است. از این رو ارائهٔ تکنیک‌هایی که به جستجوی سریع‌تر منجر شوند، بسیار با اهمیت است. یکی از این تکنیک‌ها استفاده از چند پردازنده برای اجرای الگوریتم است. متأسفانه در بسیاری مواقع نمی‌توان همان الگوریتم سریال را به سادگی توسط چند پردازنده اجرا کرد. حتی ممکن است اجرای موازی الگوریتم سریال عملکرد خوبی نداشته باشد. به این ترتیب طراحی الگوریتم‌های موازی برای جستجو اجتناب ناپذیر است.

در ادامهٔ مقاله، ابتدا در بخش ۲ اصطلاحات و مفاهیم به کار رفته در این مقاله را تعریف می‌کنیم و در بخش ۳ مروری بر کارهای انجام شده در این زمینه می‌کنیم. در بخش‌های ۴ و ۵ به ساخت موازی FS-tree و BBD-tree خواهیم پرداخت. بخش ٦ به بیان الگوریتم موازی جستجو اختصاص داده شده است و نهایتاً بخش ۷ به نتیجه گیری مقاله پرداخته است. در دو بخش ضمیمه نیز به معرفی داده‌ساختارهای اصلی این مقاله، یعنی FS-tree و BBD-tree پرداخته‌ایم.

## ۲ تعاریف

فرض کنید $P$ مجموعه‌ای از نقاط در فضای $\mathbb{R}^d$ را نشان می‌دهد. در تمام این مقاله $d$ ثابت است. ابرمستطیل مرزی $P$ که با $R(P)$ نشان می‌دهیم، کوچکترین ابرمستطیلی است که شامل همهٔ نقاط $P$ باشد. در این تعریف، منظور از ابرمستطیل، هر حاصلضرب دکارتی به صورت $R = [x_1, x'_1] \times \ldots \times [x_d, x'_d]$ در $\mathbb{R}^d$ است. حاصلضرب بازه‌های باز یک ابرمستطیل باز ایجاد می‌کند. طول $R$ در بعد $i$ با $l_i(R) = x'_i - x_i$ نشان داده می‌شود. طول‌های بیشینه و کمینهٔ $R$ با $l_{max}(R)$ و $l_{min}(R)$ نشان داده می‌شوند. نسبت طول بزرگترین ضلع به کوچکترین ضلع یک ابرمستطیل نسبت ضلعی[1] آن نامیده می‌شود. وقتی طول تمام اضلاع با هم مساوی باشند آنگاه $R$ یک ابرمکعب است که طولش برابر $l(R) = l_{min}(R) = l_{max}(R)$ می‌باشد. به طور اختصار $l_i(R(P))$ را با $l_i(P)$ نمایش می‌دهیم. یک ابرکره به شعاع $r$ مجموعه‌ای از نقاط فضای $d$ بعدی است که فاصلهٔ آنها از یک نقطه به نام مرکز، کوچکتر یا مساوی $r$ است. نماد $d(x, y)$ را به عنوان فاصلهٔ بین $x$ و $y$ به کار می‌بریم. متناظراً برای دو مجموعهٔ $X$ و $Y$، $d(X, Y)$ نشان دهندهٔ فاصلهٔ بین دو نقطهٔ $x \in X$ و $y \in Y$ است به طوریکه $d(x, y)$ به ازای این دو نقطه کمینه است.

اندازهٔ[2] یک ابرمستطیل طول بزرگترین ضلع آن است. جعبه[3] ابرمستطیلی است که نسبت ضلعی آن توسط یک مقدار ثابت محدود شده است. به طور خاص تعریف می‌کنیم یک سلول[4] را به عنوان یک جعبه یا تفاضل مجموعه‌ای دو جعبه که یکی در دیگری محاط است. بنابراین هر سلول توسط یک جعبهٔ خارجی[5] و گاهاً یک جعبهٔ داخلی[6] مشخص می‌شود. هر سلول نشان‌دهندهٔ

[1] Aspect Ratio
[2] Size
[3] Box
[4] Cell
[5] Outer Box
[6] Inner Box

نقاطی است که در داخل آن قرار دارند. سلول‌ها به صورت نواحی بسته در نظر گرفته می‌شوند و لذا نقاطی که روی مرز بین دو سلول هستند، به طور دلخواه به یکی از سلول‌ها نسبت داده می‌شوند. اندازهٔ یک سلول، اندازهٔ جعبهٔ خارجی آن است.

یک مفهوم مهم که خصوصیت جعبه‌های داخلی را مشخص می‌کند، خاصیت چسبندگی[۷][۱] نامیده می‌شود. سلولی که جعبهٔ داخلی آن با $b_I$ و جعبهٔ خارجی آن با $b_O$ نمایش داده می‌شود را در نظر بگیرید. به طور شهودی گوییم $b_I$ برای $b_O$ چسبنده است چنانچه هر وجه $b_I$ به میزان کافی از وجه متناظر $b_O$ دور باشد یا اینکه چسبیده به آن باشد. برای بیان دقیق این خاصیت دو بازهٔ بستهٔ $[x_I, y_I] \subseteq [x_O, y_O]$ را در نظر بگیرید و فرض کنید $w = y_I - x_I$ نشان‌دهندهٔ عرض بازهٔ داخلی باشد. گوییم $[x_I, y_I]$ برای $[x_O, y_O]$ چسبنده است به شرطی که هر یک از فاصله‌های بین بازهٔ داخلی و خارجی، یعنی $x_I - x_O$ و $y_O - y_I$، یا صفر باشند یا حداقل برابر $w$. جعبهٔ داخلی $b_I$ برای جعبهٔ خارجی $b_O$ چسبنده است اگر هر یک از $d$ بازهٔ $b_I$ برای بازهٔ متناظر $b_O$ چسبنده باشد. شرط معادل دیگری برای چسبندگی به این صورت است که یک جدول $۳^d$ تایی از $b_I$ ها به مرکز $b_I$ در نظر بگیرید. $b_I$ برای $b_O$ چسبنده است به شرطی که هر مستطیل در این جدول یا به طور کامل در داخل $b_O$ باشد یا در خارج آن.

در تمام مراحل الگوریتم این خاصیت که برای هر سلول، جعبهٔ داخلی برای جعبهٔ خارجی چسبنده است، حفظ می‌گردد. چسبندگی خاصیتی است که در جنبه‌های تکنیکی الگوریتم که بعداً به آنها خواهیم پرداخت، به کار می‌رود. به طور خاص این خاصیت مانع از این می‌شود که تعداد زیادی جعبهٔ داخلی، هر یک در داخل دیگری قرار بگیرند به طوری که همهٔ آنها کاملاً نزدیک مرزهای خارجی سلول مورد نظر باشند. اگر چنین شرایطی اتفاق افتد، قادر نخواهیم بود سقف بالایی برای زمان جستجو تعیین کنیم.

## ۳ کارهای انجام شده

اصولاً الگوریتم‌های جستجو بر مبنای تقسیم نقاط و ساخت یک نوع درخت جستجو بنا شده‌اند. درخت‌های جستجوی فضایی حالت تعمیم یافتهٔ همان درخت جستجوی دودویی هستند. به عبارت دیگر، درخت‌های دودویی‌ای هستند که هر گرهٔ آنها نشان دهندهٔ یک ناحیه از فضا است. این درخت‌ها به دو دسته متوازن[۸] و متناسب[۹] تقسیم می‌شوند. تقسیم یک گره در درخت‌های متناسب به صورتی است که همواره نسبت ضلعی گره‌ها از یک مقدار ثابت کمتر باشد. با استفاده از این خاصیت ثابت می‌شود، تعداد برگ‌هایی از درخت که می‌توانند با یک ابرکره به شعاع معین همپوشانی داشته باشند، کمتر از مقدار ثابتی است که مستقل از تعداد کل نقاط است. به این رابطه، محدودیت فشردگی[۱۰] گفته می‌شود. کارایی الگوریتم جستجو، رابطهٔ معکوسی با تعداد گره‌هایی که برای پیدا کردن نزدیک‌ترین همسایه ملاقات می‌شوند، دارد. بنابراین اگر بتوان سقفی برای تعداد این گره‌ها تعیین کرد، کارایی الگوریتم بهبود می‌یابد. Callahan در [۳] یک نوع تقسیم به اسم تقسیم عادل[۱۱] معرفی کرده است و بر اساس آن درختی از این نوع به نام fair split tree می‌سازد.

در مقابل در درخت‌های متوازن، تقسیم گره‌ها به صورتی است که تعداد نقاط داخلی هر فرزند ایجاد شده از یک گره، از کسر ثابتی از تعداد نقاط خود گره کمتر است. این نوع تقسیم باعث می‌شود که ارتفاع درخت به صورت $O(\log(n))$ باشد. ارتفاع درخت یکی از پارامترهای اصلی در کارایی الگوریتم‌های جستجو است. زیرا این الگوریتم‌ها ناچارند از گرهٔ ریشه تا سطح برگ‌ها، درخت را پیمایش کنند. یک گونه از این درخت‌ها kd-tree نام دارد که توسط Bently[۵] برای جستجو در فایلی از رکوردها ارائه شده است.

به خاطر سادگی kd-tree محققین بسیاری توجه خود را معطوف به آن داشته‌اند. نتیجهٔ این تحقیقات گونه‌های مختلفی از این درخت است. به عنوان مثال Yianilos[۱۰] ساختاری به اسم vp-tree را معرفی کرده است که در آن به جای استفاده از صفحات عمود بر محورها برای تقسیم یک گره، از یک نقطه استفاده می‌کند که به آن Vantage Point می‌گوید. این نقطه مرکز ابرکره‌ای است که فضا را به دو قسمت تقسیم می‌کند. این روش به خصوص در زمینهٔ پایگاه‌های داده، جذابیت زیادی دارد. مثال دیگر sr-tree[۷] است که هر گرهٔ آن توسط اشتراک یک ابرکره و یک ابرمستطیل مشخص می‌گردد. BAR-tree[۴] نیز درختی است که بر اساس همان

---

[۷] Stickness

[۸] Balanced

[۹] به این درخت‌ها اصطلاحاً Bounded Aspect Ratio گفته می‌شود.

[۱۰] Packing Constraint

[۱۱] Fair Split

kd-tree ساخته شده است، با این تفاوت که لزومی ندارد صفحات تقسیم‌کننده بر محورهای مختصات عمود باشند. بلکه می‌توان از صفحات مایل نیز استفاده کرد. به طور خاص این صفحات دارای زاویهٔ ۴۵ درجه با محورها هستند. این صفحات مایل باعث می‌شوند که دیگر نتوان به سادگی فاصلهٔ یک نقطه از یک گره را محاسبه کرد.

اگر چه داده‌ساختارهای متعدد دیگری برای تقسیم فضایی نقاط ارائه شده است ولی تقریباً همهٔ این ساختارها و الگوریتم‌های جستجوی متناظر آنها، سریال هستند. تا جایی که نویسندگان مقاله اطلاع دارند، تنها کسی که در زمینهٔ الگوریتم‌های موازی این مسأله کار کرده است، همان آقای Callahan[۲] است که یک داده‌ساختار مبتنی بر FS-tree را به صورت موازی ساخته و یک الگوریتم موازی برای جستجو در این ساختار ارائه کرده است. زمان ساخت و جستجوی الگوریتم‌های ارائه شده توسط Callahan با استفاده از $O(n)$ پردازنده در مدل CREW PRAM به صورت $O(\log n)$ است. اگرچه این زمان بهینه است ولی واقعیت این است که نماد $O$ در این روابط، وابستگی نمایی به $d$ را در خود پنهان کرده است. از طرف دیگر، الگوریتم‌های ساخت و جستجوی ارائه شده توسط وی، بسیار پیچیده هستند. در عوض داده‌ساختاری که ما در این مقاله خواهیم ساخت (BBD-tree موازی) بسیار ساده‌تر است؛ خصوصاً الگوریتم جستجوی موازی ما بسیار ساده‌تر از الگوریتم Callahan می‌باشد. اگرچه زمان ساخت داده‌ساختار ما در این مقاله، بهینه نیست.

## ۴ ساخت موازی FS-tree

برای ساخت موازی درخت کافی است مرحلهٔ تقسیم را به صورت موازی انجام دهیم. نشان می‌دهیم در هر سطح با $O(n)$ پردازنده در مدل CREW PRAM می‌توان این کار را در زمان $O(۱)$ انجام داد. از آنجایی که عمق درخت در بدترین حالت $O(n)$ است، لذا کل زمان ساخت، $O(n)$ خواهد بود.

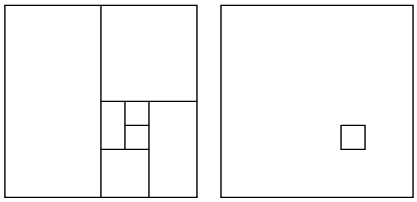
فرض کنیم در یک مرحلهٔ الگوریتم مجموعهٔ نقاط $P$ داده شده است و به هر نقطهٔ این مجموعه یک پردازنده منتسب شده است. همهٔ نقاط در یک لیست سراسری به نام $L$ ذخیره شده‌اند. به ازای هر بعد $i = ۱, \ldots, d$ یک لیست پیوندی مرتب شده از نقاط بر اساس مختصات متناظر با آن بعد می‌سازیم. این لیست‌ها که با $L_i$ نمایش می‌دهیم دارای اشاره‌گرهایی به یکدیگر هستند به طوری که از یک لیست به راحتی بتوان مکان یک نقطه در سایر لیست‌ها را پیدا کرد. اطلاعات واقعی مربوط به مختصات نقاط تنها در $L$ نگهداری می‌شود و $L_i$ها تنها حاوی اشاره‌گرهایی به مکان نقاط در این لیست سراسری هستند. با توجه به لیست‌های مرتب شدهٔ $L_i$ می‌توان مختصات ابرمستطیل حاوی نقاط را پیدا کرد. بنابراین در شروع هر مرحله لیست‌های $L_i$ حاوی نقاطی هستند که باید در بعد $s$ تقسیم شوند. همچنین ۳ تایی‌های $(l_i, m_i, h_i)$ متناظر با هر بعد $i$، نشان دهندهٔ نقاط ابتدایی، میانی و انتهایی آن بعد هستند. برای تقسیم در امتداد بعد $s$، با توجه به $(l_s, m_s, h_s)$ می‌دانیم که نقطهٔ تقسیم $m_s$ است. به ازای هر بعد $i$ دو لیست تهی جدید به نام‌های $low_i$ و $high_i$ ایجاد می‌کنیم. هر پردازندهٔ $p$ که به نقطه‌ای چون $q$ متناظر شده است، چک می‌کند که آیا $q_s < m_s$ است یا نه؟ اگر $q_s < m_s$ باشد آنگاه با توجه به اشاره‌گرهایش در $L_i$ها مکانش را در $low_i$ها به روز در می‌آورد. در غیر این صورت مکان جدیدش در $high_i$ها خواهد بود. سپس برای این $۲d$ لیست جدید، سه‌تایی‌های $(l, m, h)$ را تولید می‌کنیم. توجه کنید که فقط برای $low_s$ و $high_s$ باید مقادیر جدیدی به صورت زیر محاسبه کرد:

$$\begin{cases} (l_{low}, m_{low}, h_{low}) = (l_s, \frac{l_s+m_s}{۲}, m_s) \\ (l_{high}, m_{high}, h_{high}) = (m_s, \frac{m_s+h_s}{۲}, h_s) \end{cases}$$

پس از این مرحله لیست‌های $L_i$ قبلی حذف می‌شوند و همین فرایند به صورت بازگشتی ادامه می‌یابد. واضح است که هر مرحلهٔ الگوریتم توسط $O(n)$ پردازنده در زمان $O(۱)$ قابل انجام است. بنابر این زمان ساخت درخت در بدترین حالت $O(n)$ خواهد بود.

## ۵ ساخت موازی BBD-tree

BBD-tree بر اساس ترکیب اعمال برش و انقباض ساخته می‌شود. یادآوری می‌کنیم که یک مجموعهٔ $P$ از $n$ نقطه در $\mathbb{R}^d$ داده شده



شکل ۲: جعبهٔ انقباض برشی

است و فرض می‌کنیم که $C$ نشان دهندهٔ ابر مکعبی باشد که تمام نقاط $P$ را شامل می‌شود. ریشهٔ BBD-tree گره‌ای است که نشان دهندهٔ سلول متناظر با $C$ است و مجموعه نقاط داخلی آن، کل مجموعهٔ $P$ است. الگوریتم بازگشتی به این صورت است که یک سلول و مجموعه نقاط متناظر با آن را می‌گیرد. هر مرحلهٔ الگوریتم مشخص می‌کند که چگونه یک سلول را از طریق برش یا انقباض تقسیم کنیم. پس از تقسیم نقاط بین گره‌های فرزند همین فرایند روی فرزندان تکرار می‌شود. این فرایند تا جایی ادامه می‌یابد که تعداد نقاط متناظر با سلول جاری برابر یک باشد، در این صورت این سلول به برگی از درخت تبدیل خواهد شد. با داشتن یک گره که بیشتر از یک نقطه دارد، ابتدا تعیین می‌کنیم که عمل برش مورد نیاز است یا عمل انقباض. همان طور که قبلاً گفته شد، برش روش ارجح است زیرا هم به سادگی قابل انجام است و هم اینکه پس از به کار گیری $d$ برش پیاپی، اندازهٔ هندسی سلول متناظر، با یک فاکتور ثابت کاهش می‌یابد. اگر چه برش نمی‌تواند تضمین کند که نقاط به طور مساوی تقسیم شده‌اند ولی خواهیم دید که انقباض می‌تواند چنین خاصیتی را برای ما تضمین کند. یک روش ساده این است که برش‌ها و انقباض‌ها به طور متناوب اعمال گردند. این روش تضمین می‌کند که با پایین رفتن در درخت، هم اندازهٔ هندسی سلول‌ها و هم تعداد نقاط آنها به طور نمایی کاهش می‌یابد. یک روش عملی‌تر که ما در این مقاله استفاده می‌کنیم به این صورت است که تا وقتی با برش دادن تعداد نقاط متناظر فرزندان با یک ضریب ثابت کاهش می‌یابد، از عمل انقباض استفاده نمی‌کنیم. چنانچه این شرط نقض شود ناچاراً از عمل انقباض استفاده می‌کنیم. پس از اینکه مشخص شد باید برش انجام شود یا انقباض، صفحهٔ برش یا جعبهٔ انقباض توسط الگوریتمی که بعداً توضیح داده خواهد شد، محاسبه می‌شود. با مشخص شدن صفحهٔ برش یا جعبهٔ انقباض، این اطلاعات در سلول جاری نگهداری می‌شوند سپس دو گرهٔ فرزند ایجاد شده و به گرهٔ جاری پیوند زده می‌شوند. در نهایت مجموعهٔ نقاط، بین این دو گرهٔ فرزند تقسیم می‌شوند. چنانچه نقاطی روی مرز تقسیم باشند، این نقاط جوری به گره‌های فرزند اختصاص داده می‌شوند که مجموعه‌های حاصل تا حد امکان مساوی باشند. این کار به طور بازگشتی روی گره‌های فرزند تکرار می‌گردد.

به این ترتیب تنها قسمتی که باید موازی شود، نحوهٔ پیدا کردن جعبهٔ انقباض است. دو روش برای این کار پیشنهاد می‌شود. روش اول که «جعبهٔ انقباض برشی»[12] می‌نامیم نیازمند پیش‌ساخت یک FS-tree[۳] است. قاعدهٔ تقسیم در این درخت همان قاعدهٔ تقسیم از وسط[۱] است. روش دوم که «جعبهٔ انقباض گوشه‌ای»[13] می‌نامیم، برگرفته شده از روشی است که Callahan در ساختن داده‌ساختارش[۳] استفاده کرده است. اکنون به تشریح این دو روش می‌پردازیم.

## ۱.۵ جعبهٔ انقباض برشی

در این بخش نحوهٔ پیدا کردن جعبهٔ انقباض برشی با استفاده از یک FS-tree پیش‌ساخته شرح داده می‌شود. با توجه به الگوریتمی که در[۱] برای محاسبهٔ این نوع جعبه‌ها ذکر شده است، به سادگی می‌توان دید که تقسیمات نقطهٔ میانی و نواحی تولید شده در فرایند محاسبه همواره ثابت هستند. لذا می‌توان پیشاپیش این تقسیمات را انجام داد که حاصل آن یک FS-tree خواهد بود. حال برای پیدا کردن جعبهٔ انقباض کافی است یک بار این درخت را پیمایش کنیم. این پیمایش تا جایی ادامه پیدا می‌کند که برای اولین بار به گره‌ای برسیم که تعداد نقاطش بیشتر از ۱/۳ نقاط گرهٔ جاری باشد. به این ترتیب دو گرهٔ فرزند خواهیم داشت که هر یک از آنها حداکثر ۲/۳ نقاط را خواهند داشت. لذا ارتفاع درخت ساخته شده به صورت $\log_{\frac{3}{2}} n$ خواهد بود.

---

[12] fair-split shrinking box
[13] corner-split shrinking box

بنابر این مسألهٔ پیدا کردن جعبهٔ انقباض برشی به مسألهٔ پیمایش یک درخت دودویی با $n$ گره کاهش می‌یابد. برای پیمایش این درخت به صورت موازی می‌توان با تبدیل به مسألهٔ رتبه بندی لیستی[14] این کار را در زمان $O(\log n)$ توسط $O(n)$ پردازنده انجام داد. برای جزئیات بیشتر از نحوهٔ این تبدیل به[۹] مراجعه کنید.

## ۲.۵ جعبهٔ انقباض گوشه‌ای

این روش بر اساس لم بعد بنا شده است.

**لم ۱.** فرض کنید $R$ یک ابرمستطیل $d$ بعدی است که شامل $n$ نقطه است. فرض کنید $S$ مجموعهٔ ابرمستطیل‌های مشابه با $R$ باشد که در یک گوشه[15] با آن اشتراک دارند. فرض می‌کنیم که هیچ دو نقطه‌ای روی اضلاع یک ابرمستطیل قرار ندارند. در این صورت ابرمستطیل $R' \in S$ وجود دارد به طوری که $l_{max}(R') \leq \frac{1}{2} l_{max}(R)$ و $R'$ شامل $\lfloor \frac{n}{2^d} \rfloor$ نقطه از مجموعهٔ نقاط است.

**اثبات.** فرض کنید با نصف کردن در امتداد هر بعد، $R$ را به $2^d$ ابرمستطیل مشابه تقسیم کرده‌ایم. واضح است که یکی از این ابرمستطیل‌ها شامل حداقل $\lfloor \frac{n}{2^d} \rfloor$ نقطه از مجموعه است. این ابرمستطیل را انتخاب کرده، آن را $\widetilde{R}$ می‌نامیم. حال همهٔ ابرمستطیل‌هایی را که محاط در $\widetilde{R}$ هستند و همان گوشهٔ $R$ را با $\widetilde{R}$ به اشتراک دارند، در نظر بگیرید. همهٔ این ابرمستطیل‌ها تنها یک نقطه روی اضلاعشان دارند. از آنجایی که هیچ دو نقطه‌ای روی اضلاع یک ابرمستطیل نیستند، لذا یکی از این ابرمستطیل‌ها دقیقاً $\lfloor \frac{n}{2^d} \rfloor$ نقطه (شامل نقطهٔ روی مرز) در بر دارد. این ابرمستطیل را $R'$ می‌نایم. واضح است که $l_{max}(R') \leq \frac{1}{2}\, l_{max}(R)$. □

الگوریتم ما به این صورت است. مجموعهٔ $P$ شامل $n$ نقطه و ابرمستطیل خارجی آن، $\widetilde{R}(P)$ (حاصل از مرحلهٔ قبل، در صورتی که ریشه نباشد)، داده شده‌اند. فرض کنید $R$ ابرمستطیلی باشد که شامل $P$ است و در $\widetilde{R}$ محاط شده‌است و شرط $l_{min}(R) \geq \frac{2}{3}\, l_{max}(R)$ را برآورده می‌کند. الگوریتم تقسیم ما تضمین می‌کند که همواره چنین ابرمستطیلی وجود دارد. ابرمستطیل $R'$ را پیدا می‌کنیم طوری که شرایط لم۱ را برآورده کند. برای پیدا کردن $R'$ به یک سری پیش‌پردازش نیاز داریم. به ازای هر گوشهٔ $R$، آرایه‌ای مرتب شده از نقاط می‌سازیم. ترتیب نقاط در این آرایه‌ها نشان‌دهندهٔ ابرمستطیل‌های مشابهی است که در آن گوشه با $R$ مشترک هستند. از آنجایی که تعداد گوشه‌ها ثابت است، می‌توان با $O(n)$ پردازنده این آرایه‌ها را در زمان $O(\log(n))$ ساخت. با استفاده از این آرایه‌ها به راحتی می‌توان $R'$ و گوشهٔ متناظر را پیدا کرد. به این ترتیب دو گرهٔ فرزند ایجاد می‌شود که هر یک از آنها حداکثر $(1 - \frac{1}{2^d})$ نقاط را در بر خواهد داشت. لذا ارتفاع درخت ساخته شده به صورت $\log_{\frac{2^d}{2^d - 1}} n$ خواهد بود.

# ۶ الگوریتم جستجوی موازی

قبل از اینکه به الگوریتم اصلی بپردازیم، حالتی را در نظر می‌گیریم که در آن یک پردازنده تک نزدیک‌ترین نقطه در $P$ به یک نقطهٔ پرس‌وجو مثل $q$ را جستجو می‌کند. واضح است که این مسأله در زمان $O(n)$ توسط یک جستجوی سادهٔ brute-force قابل حل است. روش‌های متعددی پیشنهاد شده است که مقدار ثابت این رابطه را کاهش می‌دهند. تمرکز ما در این مقاله روی روش‌هایی است که ابتدا یک ساختمان دادهٔ متناظر با نقاط را در حافظه می‌سازند و سپس در آن جستجو می‌کنند. الگوریتمی که ما در این مقاله از آن استفاده می‌کنیم، اولین بار در [۵] ارائه شده است. توصیف این الگوریتم به صورت زیر است.

**الگوریتم ۱** پرس‌وجوها توسط یک الگوریتم بازگشتی جواب داده می‌شوند. در حالت پایه وقتی الگوریتم به یک برگ درخت می‌رسد، فاصلهٔ نقطهٔ پرس‌وجو را تا تمام نقاط متناظر آن برگ محاسبه می‌کند. کمترین این مقادیر به عنوان نزدیک‌ترین نقطه تا این مرحله، ذخیره می‌گردد. وقتی الگوریتم یک گرهٔ داخلی را ملاقات می‌کند، ابتدا طرفی از ابرصفحهٔ تقسیم را که نقطهٔ پرس‌وجو در آن است، مشخص می‌کند. سپس جستجو به صورت بازگشتی روی این فرزند اجرا می‌گردد. در بازگشت از جستجو مشخص می‌کند که

[14] List Ranking
[15] Corner

آیا فرزند دیگر این گره، نزدیک‌تر از نقطهٔ پیدا شده تا این مرحله، به نقطهٔ پرس‌وجو است؟ اگر چنین باشد این فرزند نیز به طور بازگشتی ملاقات می‌گردد. وقتی جستجو به ریشه بازگردد، نزدیک‌ترین نقطه‌ای که پیدا شده است، بر گردانده می‌شود.

یک مشاهدهٔ مهم در این الگوریتم این است که به ازای هر نقطهٔ پرس‌وجو، هر برگی که فاصله‌اش از نقطهٔ پرس‌وجو کمتر از نزدیک‌ترین همسایه است، توسط الگوریتم ملاقات خواهد شد. به عبارت دیگر اگر فرض کنیم فاصلهٔ نزدیک‌ترین همسایه تا نقطهٔ پرس‌وجو $r$ باشد، همهٔ برگ‌هایی که با یک ابرکرهٔ به شعاع $r$ همپوشانی دارند، توسط این الگوریتم ملاقات می‌شوند. از طرف دیگر با توجه به خاصیت محدودیت فشردگی BBD-tree که در لم۳ بیان شده است، تعداد این برگ‌ها مستقل از $n$ می‌باشد.

اینک به توصیف الگوریتم موازی می‌پردازیم.

**الگوریتم ۲** در این حالت $n$ پردازنده موجود است که هر یک موظف به پیدا کردن نزدیک‌ترین همسایه برای یکی از $n$ نقطهٔ مجموعهٔ $P$ است. لذا هر پردازنده همان الگوریتم۱ را به طور مستقل اجرا می‌کند. مراحل الگوریتم به این صورت است:

۱) یک BBD-tree متناظر با نقاط ساخته می‌شود.

۲) هر نقطه به یک پردازنده منتسب می‌گردد.

۳) هر پردازنده الگوریتم۱ را اجرا می‌کند.

**لم ۲ .** زمان اجرای الگوریتم موازی $O(\log n)$ است.

**اثبات.** واضح است که زمان اجرای این الگوریتم همان زمان اجرای الگوریتم۱ توسط یک پردازنده است. پیمایش درخت تا سطح برگ در زمان $O(\log n)$ انجام می‌شود. چنانچه تعداد برگ‌هایی که توسط الگوریتم ملاقات می‌شوند مستقل از $n$ (یعنی کمتر از یک مقدار ثابت) باشد، لذا زمان اجرای الگوریتم۱ و در نتیجه الگوریتم۲ به صورت $O(\log n)$ خواهد بود. نشان می‌دهیم که تعداد این برگ‌ها کمتر از $\lceil ۱ + ٦d\rceil^d$ است.

فرض کنید یک پردازنده به نقطهٔ $q$ منتسب شده است و همچنین فرض کنید $r$ نشان دهندهٔ فاصلهٔ $q$ تا آخرین برگی باشد که باعث خاتمهٔ الگوریتم نشده است. می‌دانیم که تمام گره‌هایی که تا کنون ملاقات شده‌اند در فاصلهٔ $r$ از $q$ هستند. اگر $p$ نزدیک‌ترین نقطهٔ پیدا شده تا این مرحله باشد، چون الگوریتم خاتمه نیافته پس

$$r \leq dist(q, p)$$

ادعای ما این است که هیچ گرهٔ ملاقات شده‌ای وجود ندارد که اندازهٔ آن کمتر از $r/d$ باشد. فرض کنید چنین گره‌ای ملاقات شده باشد. این گره در فاصلهٔ $r$ از $q$ قرار دارد و لذا با ابرکره‌ای که به مرکز $q$ و به شعاع $r$ باشد همپوشانی خواهد داشت. در هر معیار Minkowski قطر این گره نمی‌تواند از $d$ برابر طولانی‌ترین ضلعش بیشتر باشد و لذا کمتر از $r$ خواهد بود. از آنجایی که این گره متناظر یک نقطه است، الگوریتم جستجو می‌بایست به نقطه‌ای با فاصلهٔ کمتر از $r$ برخورد کرده باشد. در حالی که این خلاف فرضی است که $p$ نزدیک‌ترین نقطه‌ای است که تا کنون دیده شده است.

بنابراین تعداد گره‌هایی که تا قبل از خاتمه توسط الگوریتم ملاقات می‌شوند محدود به گره‌های با اندازهٔ بزرگتر از $r/d$ است که با یک ابرکرهٔ به شعاع $r$ همپوشانی دارند. با توجه به لم۳ این تعداد مستقل از $n$ و حداکثر برابر $\lceil ۱ + ٦d\rceil^d$ است. □

به طور ضمنی فرض کرده‌ایم که محاسبهٔ فاصلهٔ نقطهٔ پرس‌وجو تا یک گرهٔ درخت توسط یک پردازنده در زمان $O(d)$ قابل انجام است.

# ۷ نتیجه گیری

در این مقاله الگوریتمی با زمان $O(\log^{۲}(n))$ برای ساخت موازی یک BBD-tree متناظر با یک مجموعهٔ $P$ شامل $n$ نقطه در فضای $\mathbb{R}^d$، با $O(n)$ پردازنده در مدل CREW PRAM ارائه گردید. سپس بر اساس آن یک الگوریتم موازی برای جستجوی نزدیک‌ترین همسایه‌های نقاط $P$ ارائه شد که زمان جستجوی آن در درخت از مرتبهٔ $O(\log(n))$ می‌باشد.

از جمله کارهای دیگری که می‌توان در این زمیه انجام داد، بهینه کردن مرحلهٔ ساخت درخت است. در حالتیکه از برش‌های ساده برای پیدا کردن جعبهٔ انقباض استفاده کنیم نیازمند پیش‌سازی یک FS-tree هستیم که به نظر می‌رسد بتوان ساختار مشابهی را در زمان $O(\log n)$ ساخت. همین طور در پیدا کردن گوشهٔ انقباض ممکن است با انجام پیش‌پردازش‌هایی بتوان به زمان $O(۱)$ دست یافت. پیاده‌سازی الگوریتم و کاهش تعداد پردازنده‌ها به تعداد $O(n/\log(n))$ یا در حالت کلی $P$ پردازنده، از جمله کارهای جالبی است که می‌توان در این زمینه انجام داد.

# مراجع


[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of ACM*, 45:891-923, 1998.

[2] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th Sympos. on Foundations of Computer Science*, pages 332-341, 1993.

[3] P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potentional fields. *Journal of ACM*, 42:67-90, 1995.

[4] C. Duncan, M. Goodrich and S. Kobourov. Balanced aspect ratio trees: Combining the advantages of kd-trees and octrees. In *Proc. 10th ACM-SIAM Sympos. on Discrete Algorithms*, pages 300-309, 1999.

[5] J. H. Friedman, J. L. Bentley and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software*, 3:209-226, 1977

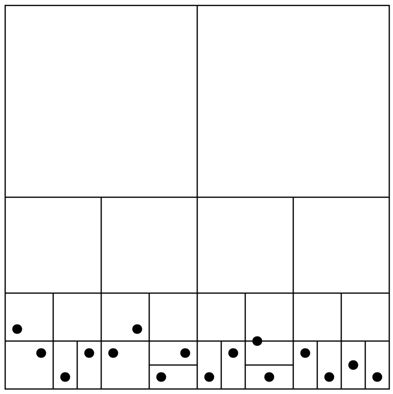
[6] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman and A. Wu. An efficient k-means clustering algorithm: Analysis and Implementation. In *Proc. 16th ACM Sympos. on Computational Geometry*, 2000.

[7] N. Katayama and S. Satoh. The SR-tree: An index structure for high dimensional nearest neighbor queries. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 369-380, 1997.

[8] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. 29th STOC*, 1997.

[9] M. J. Quinn. *Parallel Computing*. second edition, Mc-GRAW HILL Inc., 1994.

[10] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Sympos. on Discrete Algorithms*, pages 311-321, 1993.



شکل ۳: نحوهٔ تقسیم فضا در FS-tree

## A FS-tree

در این قسمت به معرفی درختی به نام Fair Split Tree می‌پردازیم. این داده‌ساختار اولین بار در[۳] مطرح گردیده است. البته درختی که ما در اینجا معرفی می‌کنیم تا حدی ساده‌تر از آنچه در آنجا بیان شده است، می‌باشد. به طور خاص این سادگی از آنجا ناشی شده است که ما فرض می‌کنیم همواره ابرصفحهٔ تقسیم از وسط ضلع تقسیم عبور می‌کند. این ساختار در واقع شکلی از kd-tree است که با قاعدهٔ تقسیم از وسط ساخته می‌شود. استفاده از نام FS-tree تنها به خاطر شباهت با داده‌ساختار معرفی شده در[۳] می‌باشد. از نظر دسته بندی، این درخت در گروه ساختارهای متناسب قرار می‌گیرد. فرض کنیم مجموعه نقاط $P$ داده شده است. یک برش $P$ را تعریف می‌کنیم به عنوان تقسیم آن به دو مجموعهٔ ناتهی که هر کدام در یک طرف ابرصفحهٔ (به نام ابرصفحهٔ تقسیم) عمود بر یکی از محورهای مختصات، قرار دارند. یک درخت برش (split tree) متناظر با مجموعه نقاط $P$، یک درخت دودویی است که به صورت بازگشتی ساخته می‌شود. فرایند ساخت به این صورت است که اگر $|P| = ۱$ آنگاه مجموعهٔ $P$ دارای یک درخت برش منحصر به فرد است که شامل گرهٔ $P$ است. در غیر این صورت درختی خواهد بود که ریشهٔ آن $P$ است و دارای دو زیر درخت است که هر یک درخت برش متناظر با یکی از دو زیرمجموعهٔ حاصل از برش $P$ می‌باشد. به ازای هر گرهٔ $A$ در درخت، پدر این گره را (در صورت وجود) با $p(A)$ نشان می‌دهیم.

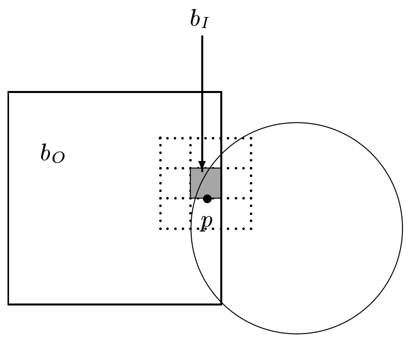
ابرمستطیل خارجی متناظر با گرهٔ $A$ را که با $\widetilde{R}(A)$ نمایش می‌دهیم، به صورت بالا به پایین تعریف می‌کنیم:

- برای ریشهٔ $P$، $\widetilde{R}(P)$ نشان دهندهٔ ابرمکعبی است که مرکزش منطبق بر مرکز $R(P)$ است و $l(\widetilde{R}(P)) = l_{max}(P)$.
- برای سایر گره‌ها مثل $A$، ابر صفحهٔ تقسیمی که برای تقسیم $p(A)$ به کار گرفته شده است، $\widetilde{R}(p(A))$ را نیز به دو ابرمستطیل باز تقسیم می‌کند. $\widetilde{R}(A)$ ابرمستطیلی است که شامل $A$ است.

یک fair split گرهٔ $A$ را به عنوان برشی از $A$ تعریف می‌کنیم که در آن ابرصفحهٔ تقسیم از وسط وجه تقسیم می‌گذرد و موازی با دو وجه متناظر $\widetilde{R}(A)$ است. درخت برشی که تنها با استفاده از برش‌های عادل ساخته می‌شود، fair split tree یا به اختصار FS-tree نامیده می‌شود.

## B BBD-tree

در این بخش به معرفی درختی به نام Balanced Box Decomposition Tree یا به اختصار BBD-tree می‌پردازیم. این داده‌ساختار اولین بار در [۱] مطرح گردید. این درخت یکی از داده‌ساختارهای هندسی است که بر اساس تقسیم سلسله مراتبی فضا به مستطیل‌هایی با اضلاع عمود بر محورهای مختصات، ساخته می‌شود. خصوصیت اصلی BBD-tree این است که دو ویژگی مهم این داده‌ساختارها را در یک داده‌ساختار فراهم کرده است. به عنوان مثال kd-tree[۵] را در نظر بگیرید. این داده‌ساختار به طور بازگشتی توسط ابرصفحهٔ عمود بر محورها، فضا را طوری تقسیم می‌کند که دو ناحیهٔ ایجاد شده شامل تعداد نقاط تقریباً مساوی باشند. در نتیجه همین طور که در درخت به پائین حرکت می‌کنیم، کاردینالیتی نقاط داخلی گره‌های مسیر به صورت نمایی کاهش می‌یابد. در




شکل ۴: محدودیت فشردگی

مقابل داده‌ساختارهای مبتنی بر quad-tree[۳] را در نظر بگیرید که در آنها، فضا به نواحی مستطیلی با نسبت ضلعی محدود تقسیم می‌شود. در این سری ساختارها با پائین رفتن در درخت اندازهٔ هندسی هر گره (اندازهٔ بزرگترین ضلع) به طور نمایی کاهش می‌یابد.

BBD-tree بر اساس یک نوع تقسیم فضایی ساخته می‌شود که هم کاهش نمایی کاردینالیتی و هم کاهش نمایی اندازه را شامل می‌شود. تفاوت اصلی بین BBD-tree و سایر داده‌ساختارهای فوق این است که هر گرهٔ BBD-tree یک ابرمستطیل ساده نیست، بلکه در حالت کلی تفاضل مجموعه‌ای دو ابرمستطیل است که یکی در دیگری محاط است. واضح است که می‌توان این نواحی را توسط برش‌هایی به $۲d$ ابرمستطیل تبدیل کرد؛ البته ابرمستطیل‌های حاصل ممکن است نسبت ضلعی محدودی نداشته باشند. یک BBD-tree دارای $O(n)$ گره است، لذا حافظهٔ مورد نیاز برای نگهداری آن از مرتبهٔ $O(dn)$ خواهد بود.

BBD-tree با به کارگیری مکرر دو عمل برش عادل (یا به سادگی برش) و انقباض[۱۶] ساخته می‌شود. این دو عمل به طور شهودی، روش‌هایی هستند برای تقسیم یک سلول به سلول‌های کوچکتر که فرزندان آن نامیده می‌شوند. این فرزندان با توجه به اینکه مختصاتشان کمتر یا بیشتر از صفحهٔ تقسیم باشد، فرزند پایین[۱۷] و بالا[۱۸] نامیده می‌شوند. عمل انقباض یک سلول را به دو سلول تقسیم می‌کند ولی به جای استفاده از صفحهٔ تقسیم از یک جعبه (که جعبهٔ انقباض نامیده می‌شود) استفاده می‌کند. این عمل یک سلول را به دو فرزند تقسیم می‌کند که یکی در داخل و دیگری در خارج قرار دارد. هر دو عملیات طوری انجام می‌شوند که خصوصیات زیر همواره برقرار باشند:

۱) همهٔ جعبه‌ها خاصیت محدودیت نسبت ضلعی را برآورده سازند.

۲) اگر سلول پدر دارای یک جعبهٔ داخلی است، پس از تقسیم، این جعبه به طور کامل در یکی از فرزندان قرار خواهد داشت. اگر این عمل انقباض باشد، آنگاه این جعبهٔ داخلی در داخل فرزند داخلی قرار خواهد داشت.

۳) جعبه‌های داخلی برای جعبه‌های خارجی متناظرشان چسبنده باشند.

واضح است که وقتی فقط برش‌ها اعمال می‌شوند، در حالت کلی نمی‌توان تضمین کرد که نقاط به صورت مساوی تقسیم می‌شوند لذا درختی که فقط در نتیجهٔ برش‌ها ساخته می‌شود، ممکن است متوازن نباشد. به طور شهودی عمل انقباض باعث می‌شود تا سریعاً به نواحی‌ای برسیم که چگالی نقاط در آنها بالا است. توجه کنید که برش حالت خاصی از انقباض است که در آن جعبهٔ انقباض $۲d - ۱$ ضلع مشترک با جعبهٔ خارجی دارد. تفاوتی که از نقطه نظر عملی وجود دارد این است که برای مشخص کردن اینکه یک نقطه در داخل جعبهٔ انقباض قرار دارد به $۲d$ مقایسه نیاز است در حالی که پیدا کردن طرفی از صفحهٔ برش که نقطهٔ مورد نظر در آن است تنها نیازمند یک مقایسه است.

خاصیت اساسی BBD-tree که ما از آن استفاده می‌کنیم، همان محدودیت فشردگی است که در لم بعد بیان می‌شود.

**لم ۳.** یک BBD-tree متناظر با یک مجموعه نقطه در $\mathbb{R}^d$ موجود است. تعداد برگ‌های با حداقل اندازهٔ $s$ که با یک ابرکرهٔ با شعاع $r$ تحت معیار $L_m$ همپوشانی دارند، حداکثر برابر $\lceil ۶r/s + ۱\rceil^d$ است.

---

[۱۶] shrink
[۱۷] low
[۱۸] high