هندسه محاسباتی پیشرفته

مساله نزدیک‌ترین همسایه متنوع

سپیده آقاملائی

۲۰ بهمن ۱۳۹۲

چکیده

گزارش کردن k نزدیک‌ترین همسایه متنوع در جستجوی آگاه نسبت به تنوع کاربرد دارد. تعریف مساله به این صورت است که یک نقطه‌ی q داده شده است، مجموعه S شامل k نقطه درون توپ به شعاع r با بیشترین تنوع را گزارش کنید. تنوع یک مجموعه مثل S با حداقل فاصله بین دو نقطه‌ی S سنجیده می‌شود که هرچه بیشتر باشد بهتر است.
در این مقاله دو الگوریتم تقریبی برای حالتی که نقاط درون یک فضای فاصله‌ی همینگ d-بعدی قرار دارند، ارائه می‌شود. زمان کوئری الگوریتم خطی نسبت به n و چندجمله‌ای بر حسب پارامتر تنوع k و تعداد ابعاد d است. برای مقادیر کم k، الگوریتم زمان کوئری خطی دارد، حتی اگر تعداد نقاط با فاصله‌ی r از نقطه کوئری q برحسب n خطی باشند. این الگوریتم‌ها اولین الگوریتم‌هایی از این نوع هستند که پیچیدگی قابل اثبات دارند.

۱ مقدمه

مساله‌ی گزارش همسایه‌های نزدیک (یا جستجوی بازه‌ای) به این صورت تعریف می‌شود: مجموعه P از n نقطه داده شده است، ساختمان داده‌ای بسازید که به ازای هر نقطه کوئری داده شده، همه‌ی نقاط داده با فاصله کمتر از r از نقطه کوئری را گزارش کند.
این مساله در زمینه‌های مختلفی از جمله پایگاه داده، داده‌کاوی، ذخیره و بازیابی اطلاعات، پایگاه داده‌های تصویر و فیلم، تشخیص الگو، آمار و تحلیل داده‌ها کاربرد دارد. در این کاربردها ویژگی‌های هر شئ مورد بحث (متن، تصویر،...) به صورت یک نقطه در فضای d بعدی بازنمایی می‌شوند و ملاک فاصله برای اندازه‌گیری شباهت اشیا به کار می‌رود. در نتیجه مساله اصلی اندیس‌گذاری یا جستجوی معنایی برای اشیای کوئری است. تعداد ویژگی‌ها (ابعاد) از ده تا هزاران متغیر است.
یکی از مشکلات اصلی جستجوی معنایی تعداد جواب‌هایی است که بازیابی و گزارش می‌شوند. اگر اندازه

مجموعه جواب خیلی کوچک باشد (مثلاً تنها چند نقطه که به کوئری نزدیک‌تر اند)، جواب می‌تواند بسیار همگن و فاقد اطلاعات مفید باشد. اگر تعداد نقاط گزارش شده خیلی زیاد باشد، زمان بازیابی آنها زیاد می‌شود. همچنین جواب‌های بزرگ هم خیلی اطلاعات مفیدی نمی‌دهند. در چند سال اخیر، این مشکل هدف تحقیقات زیادی روی جستجوهای آگاه نسبت به تنوع بوده‌است. هدف این کار طراحی الگوریتم‌های کارا برای بازیابی جوابهای مرتبط (نزدیک به نقطه کوئری) و متنوع است. تنوع می‌تواند به شکل‌های مختلفی تعریف شود. یکی از روش‌های مشهور خوشه‌بندی جوابها و برگرداندن مرکزها است. این روش می‌تواند زمان اجرای زیاد داشته باشد اگر نقاط تعدادشان زیاد باشد.

۱.۱ نتایج ما

در این مقاله دو الگوریتم تقریبی کارا برای مساله k *همسایه نزدیک متنوع* داده می‌شود. مساله به این شکل تعریف می‌شود: به ازای یک نقطه کوئری داده شده، مجموعه *نقاط متنوع* S از k نقطه را در توپ با شعاع r حول q گزارش کنید. تنوع مجموعه S کمینه‌ی فاصله‌ی بین هر دو نقطه‌ی S است. به عبارت دیگر، الگوریتم جواب تقریبی مساله خوشه‌بندی k-مرکز روی لیست نقاط نزدیک به کوئری را گزارش می‌کند. زمان اجرا، ضریب تقریب و فضای حافظه‌ی الگوریتم در جدول داده شده‌اند. توجه کنید الگوریتم A از الگوریتم B بدتر است اما الگوریتم ساده‌تری است و پیاده‌سازی و تحلیل آن هم ساده‌تر است و قبلاً از آن برای بازیابی اخبار متنوع استفاده شده است.
ویژگی‌های اصلی الگوریتم ما تضمین زمان کوئری برای خطی بودن بر حسب n و چندجمله‌ای بودن بر حسب پارامتر تنوع k و تعداد ابعاد d است، در حالی که ضریب تقریب ثابت برای رسیدن به تنوع تامین شود. توجه کنید برای مقادیر کم k الگوریتم ما زمان کوئری خطی دارد، حتی اگر تعداد نقاط به فاصله‌ی حداکثر r از q بر حسب n خطی باشند. توجه کنید تقریب زدن تنوع اجتناب ناپذیر است چون پیدا کردن مجموعه با اندازه‌ی k که تنوع را بیشینه کند با ضریب کمتر از ۲، NP-سخت است. اینها اولین الگوریتمهایی از این نوع هستند که تضمین قابل اثبات می‌دهند. یکی از الگوریتم‌های ما (الگوریتم A) به الگوریتم‌هایی که در عمل برای این کار استفاده می‌شوند نزدیک است. هرچند این مقالات تضمینی برای کیفیت جواب نداده‌اند.

۲.۱ کارهای گذشته

در این قسمت مروری از کارهای گذشته روی تقریب همسایه نزدیک و جستجوی آگاه نسبت به تنوع که به نتایج این مقاله مرتبط هستند، انجام می‌دهیم.
همسایه نزدیک.
مساله همسایه نزدیک موضوع تحقیقات بسیاری بوده است. چندین الگوریتم کارا برای وقتی که تعداد ابعاد d کم باشد وجود دارد. با این وجود جوابهای فعلی یا زمان کوئری یا حافظه‌ی نمایی بر حسب d دارند. بنابرین در سالهای اخیر روشهای تقریبی برای از بین بردن این تنگنا پیشنهاد شده‌اند. در تقریب گزاش همسایه نزدیک/ جستجوی بازه‌ای، الگوریتم باید همه‌ی نقاط در فاصله r از q را برگرداند و می‌تواند تعدادی نقطه در فاصله cr از q را هم برگرداند.
یکی از روشهای مشهور برای نزدیک‌ترین همسایه در ابعاد بالا بر مبنای مفهوم locality sensitive hashing (LSH) است. ایده هش کردن نقاط توسط تعدادی (L) تابع هش است تا مطمئن شویم برای هر تابع، احتمال برخورد برای نقاط نزدیک به هم بسیار بیشتر از نقاط دور از هم است. سپس می‌توان گزارش همسایه نزدیک را با هش کردن نقطه‌ی کوئری و بازیابی تمام اعضایی که در سطل‌های شامل آن نقطه هستند؛ حل کرد. این روش مثلا در بسته‌ی E2LSH برای جستجوی معنایی ابعاد بالا استفاده شده

است.
الگوریتم LSH بر حسب تابع‌های فاصله‌ی به کار رفته در آن انوع مختلفی دارد. در ساده‌ترین حالت وقتی عدم شباهت بین نقاط کوئری با فاصله‌ی همینگ تعریف شود، الگوریتم تضمین ۱) هر نقطه در فاصله کمتر از r از q با یک احتمال ثابت (قابل تنظیم) گزارش می‌شود. ۲) زمان کوئری حداکثر $O(d(n^{1/c} + |P_{cr}(q))$ است که $P_R(q)$ مجموعه نقاطی از P را مشخص می‌کند که تا q فاصله‌ی R دارند. پس اگر اندازه‌ی مجموعه جواب $P_{cr}(q)$ بزرگ باشد، کارایی الگوریتم کاهش پیدا می‌کند. به صورت اکتشافی افزایش سرعت می‌تواند با خوشه‌بندی نقاط در هر سطل و نگه داشتن فقط مرکز خوشه‌ها انجام شود. هر چند الگوریتم حاصل هیچ تضمینی نداشته است. (تا کنون) تنوع.
در این مقاله ما تعریف $content - based$ برای تنوع را به کار بردیم. این روش گزارش کردن k جواب است که به مقدار کافی از هم متفاوت هستند. این به صورت بیشینه کردن کمترین فاصله بین هر زوج جواب است یا میانگین فاصله بین جواب‌ها و غیره. در این مقاله ما از کمترین فاصله برای بیان ریاضی استفاده می‌کنیم و از الگوریتم خوشه‌بندی حریصانه برای پیدا کردن k خوشه‌بندی با بیشترین تنوع نقاط در یک مجموعه داده شده استفاده می‌کنیم.
تنها کار قبلی که مستقیماً تعریف ما از k همسایه نزدیک متنوع را ارائه داده است (۱) است. این مقاله الگوریتمی (مشابه الگوریتم A در این مقاله ارائه می‌دهد، اما از اندیس جاکارد به جای فاصله همینگ استفاده می‌کند) و آن را روی مسائلی در بازیابی اخبار اجرا می‌کند. هرچند این مفاله هیچ تضمینی روی دقت جوابهای گزارش شده نمی‌دهد.

۳.۱ تکنیک ما

هر دوی الگوریتم‌هایی که ما استفاده می کنیم از LSH به عنوان پایه استفاده می‌کنند. مشکل اصلی کاهش وابستگی زمان کوئری به اندازه‌ی مجموعه‌ی $P_{cr}(q)$ از نقاط نزدیک به q است. الگوریتم اول (A) به این صورت این مشکل را حل می‌کند که فقط k متنوع‌ترین نقاط را در هر سطل نگه می‌دارد. این کار تضمین می‌کند کل تعداد نقاطی که در زمان هر کوئری بررسی می‌شوند حداکثر $O(kL)$ است که L تعداد توابع هش است. اما اثبات تضمین‌های تقریب برای این الگوریتم نیازمند این است که هیچ داده‌ی دورافتاده‌ای (نقطه‌ای با فاصله‌ی بیشتر از cr از q) در هیچ سطلی ذخیره نشده باشد. در غیر این صورت این نقطه می‌تواند به عنوان یکی از k نقطه‌ی متنوع از آن سطل گزارش شود و جایگزین یک نقطه‌ی معتبر شود. این نیاز باعث می‌شود الگوریتم تنها در صورتی کار کند که ضریب تقریب فاصله (c) از ۲ بزرگتر باشد.
تضمین ضریب تقریب ۶ برای تنوع با استفاده از مفهوم مجموعه‌های هسته نشان داده شده‌است. به سادگی می‌فهمیم بیشینه‌ی k-متنوع بودن یک مجموعه نقطه کمتر از ۲ برابر هزینه‌ی جواب بهینه‌ی خوشه‌بندی $(k-1)$-مرکز آن است. برای مساله‌ی دومی روش ساختن زیرمجموعه‌از نقاط ورودی (یک مجموعه هسته) شناخته شده است که برای هر مجموعه‌ای از مرکزهای خوشه‌ها هزینه‌ی خوشه‌بندی مجموعه هسته، ضریب ثابتی از جواب بهینه‌ی خوشه‌بندی کل داده‌ها باشد. الگوریتم ما سپس به سادگی تنها یک مجموعه‌ی هسته به ازای هر سطل LSH محاسبه و ذخیره می‌کند. مشخصات استاندارد مجموعه هسته نتیجه می‌دهند که اجتماع مجموعه‌های هسته‌ی سطل‌های متناظر یک نقطه کوئری q یک مجموعه‌ی هسته برای همه‌ی نقاط این سطل‌ها است. پس اجتماع همه‌ی مجموعه‌های هسته اطلاعات کافی برای بازیابی یک جواب تقریبی بهینه برای همه‌ی نقاط نزدیک به q می‌دهد.
برای به دست آوردن الگوریتمی که برای $c > ۱$ دلخواه کار کند، باید الگوریتم ما نسبت به داده‌های دور افتاده حساس نباشد ($robust$ باشد). تحلیل روش LSH استانداردتضمین می‌کند که تعداد نقاط دورافتاده در همه‌ی سطل‌ها حداکثر $O(L)$ است. الگوریتم B به ویژگی $robustness$ با نگهداری یک مجموعه هسته‌ی $robust$ دست پیدا می‌کند که می‌تواند تعدادی از داده‌های دور افتاده را تحمل کند. چون ما از قبل نمی‌دانیم چه تعداد داده‌ی دورافتاده‌ای در یک سطل داده‌شده هست، الگوریتم ما دنباله‌ای از نقاط

را نگه می‌دارد که یک مجموعه هسته‌ی $robust$ نسبت به تعداد داده‌های دورافتاده‌ی در حال افزایش را بازنمایی می‌کنند. در زمان کوئری زدن، الگوریتم لیست را پیمایش می‌کند تا نقاط کافی برای اطمینان از $robustness$ خوانده شده باشند.

۲ تعریف مساله

فرض کنید $(\Delta, dist)$ یک فضای متریک d بعدی باشد. با دو تعریف زیر شروع می‌کنیم.

تعریف ۱ *به ازای هر مجموعه $S \in \Delta$ داده شده، تنوع به صورت کمینه فاصله‌ی بین هر دو نقطه از نقاط مجموعه تعریف می‌شود:*

$$div(S) = min_{p,p' \in S} dist(p, p')$$

تعریف ۲ *به ازای هر مجموعه $S \in \Delta$ داده شده، k-تنوع آن بیشینه‌ی تنوع قابل دستیابی با انتخاب یک زیرمجموعه k عضوی است:*

$$div_k(S) = max_{S' \subset S, |S'|=k} div(S')$$

ما همچنین به مجموعه‌ی S' که مقدار بیشینه‌ی رابطه‌ی بالا را می‌دهد k-زیرمجموعه بهینه‌ی S می‌گوییم. توجه کنید k-تنوع در حالتی که $k > |S|$ باشد تعریف نشده است.

برای اجتناب از k-تنوع مجموعه‌های با اعضای کمتر از k، قرارداد می‌کنیم هر نقطه‌ی p از مجموعه نقاط ورودی P را k بار تکرار کنیم. این کار باعث می‌شود همه‌ی مجموعه‌های ناتهی S که در ادامه‌ی مقاله آمده‌اند، کیفیت $div_k(S)$ خوش‌تعریف و اگر تعداد نقاط کمتر از k باشد، ۰ داشته باشند. به سادگی می‌فهمیم که این کار کرانهای حافظه‌ی الگوریتم را تغییر نمی‌دهد.
مساله‌ی همسایه نزدیک k-متنوع به این صورت تعریف می‌شود: به ازای نقطه‌ی کوئری داده شده‌ی q، مجموعه‌ی S را طوری گزارش کنید که: ۱) $S \in P \cap B(q, r)$ باشد که $B(q, r) = p | dist(p, q) \leqslant r$ توپ با شعاع r و مرکز q است. ۲) $|S| = k$ ۳) $div(S)$ بیشینه باشد.
به دلیل اینکه الگوریتم ما تقریبی است، باید *مساله‌ی همسایه نزدیک k-متنوع تقریبی* را تعریف کنیم. در این حالت باید برای یک ضریب تقریب $c > ۱$ و $\alpha > ۱$ داشته باشیم: ۱) $S \subset P \cap B(q, cr)$ ۲) $|S| = k$ ۳) $div(S) \geqslant \frac{۱}{\alpha} div_k(P \cap B(q, r))$.

۳ پیشنیازها

۱.۳ الگوریتم GMM

فرض کنید مجموعه نقاط $S \subset \Delta$ را داریم و می‌خواهیم یک k-زیرمجموعه‌ی بهینه برای S محاسبه کنیم. یعنی مجموعه‌ای از k نقطه پیدا کنیم که فاصله‌ی دو به دوی آنها بیشینه شود. هرچند این مساله np-سخت است، یک الگوریتم ساده‌ی ۲-تقریبی حریصانه به نام GMM برای آن وجود دارد.
در این مقاله ما از یک نسخه‌ی کمی تغییریافته‌ی GMM استفاده می‌کنیم(اثبات ضریب تقریب این نسخه دقیقاً مثل همان اثبات اصلی است). به الگوریتم یک مجموعه نقاط S و یک پارامتر k به عنوان ورودی داده می‌شود. ابتدا الگوریتم یک نقطه‌ی دلخواه $a \in S$ انتخاب می‌کند. سپس به صورت تکراری نقطه‌ی بعدی را به مجموعه خروجی اضافه می‌کند تا k نقطه داشته باشد. به طور دقیقتر، در هر قدم، به صورت

حریصانه نقطه‌ای که کوتاهترین فاصله‌اش تا نقاط انتخاب شده فعلی بیشینه باشد، اضافه می‌کند.
توجه کنید که تغییر اینکه همه‌ی نقاط k بار تکرار شوند باعث می‌شود که اگر مجموعه‌ی ورودی کمتر از k نقطه متمایز داشته باشد، مجموعه‌ی خروجی S' شامل همه‌ی نقاط باشد.

الگوریتم ۱ GMM

۱: کامنت کار نمی کرد:
۲: ورودی S: مجموعه نقاط، k: اندازه‌ی زیرمجموعه
۳: خروجی S': زیرمجموعه‌ای از S با اندازه‌ی k
۴: یک نقطه‌ی دلخواه: $S' \leftarrow a$
۵: به ازای i از ۲ تا k:
۶: $p \in S - S'$ پیدا کن که $min_{x \in S'} dist(p, x)$ را بیشینه کند.
۷: $S' \leftarrow S' \cup \{p\}$
۸: S' را برگردان.

لم ۱ *زمان اجرای الگوریتم $O(k.|S|)$ است و ضریب تقریب حداکثر ۲ برای k-تنوع $div_k(S)$ دارد.*

۲.۳ مجموعه‌های هسته

تعریف ۳ *فرض کنید $(P, dist)$ یک متریک باشد. برای هر زیرمجموعه ای از نقاط $S, S' \subset P$ ، هزینه‌ی k-مرکز را به صورت زیر تعریف می‌کنیم:*

$$KC(S, S') = max_{p \in S} min_{p' \in S'} dist(p, p')$$

مساله‌ی k-مرکز متریک به این صورت تعریف می‌شود: S داده شده‌است، زیرمجموعه‌ی $S' \in S$ با اندازه‌ی k پیدا کنید که $KC(S, S')$ را کمینه کند. ما این هزینه‌ی بهینه را با $KC_k(S)$ نشان می‌دهیم.

لم ۲ *k-تنوع یک مجموعه S ارتباط نزدیکی با هزینه‌ی بهترین $(k-۱)$-مرکز S دارد. یعنی:*

$$KC_{k-۱}(S) \leqslant div_k(S) \leqslant ۲KC_{k-۱}(S)$$

اثبات. برای نامساوی اول فرض کنید S' k-زیرمجموعه بهینه‌ی S باشد. همچنین $a \in S'$ را یک نقطه‌ی دلخواه و $S'- = S' - a$. سپس برای هر نقطه‌ی $b \in S - S'$ ، داریم $min_{p \in S'-} dist(b, p) \leqslant min_{p \in S'-} dist(a, p)$ در غیر این صورت b انتخاب بهتری از a بوده است، یعنی $div(b \cup S'-) > div(S')$. پس $KC(S, S'-) \leqslant div_k(S)$ است و نامساوی برقرار است.
برای قسمت دوم، فرض کنید $C = \{a_۱, ..., a_{k-۱}\}$ یک مجموعه بهینه از $(k-۱)$-مرکز برای S باشد. سپس چون S' اندازه‌ی k دارد، طبق اصل لانه‌کبوتری، $p, p' \in S'$ و a وجود دارند که

$$a = argmin_{c \in C} dist(p, c) = argmin_{c \in C} dist(p', C)$$

و درنتیجه طبق نامساوی مثلث داریم:

$$div_k(S) = div(S') \leqslant dist(p, p') \leqslant dist(p, a) + dist(a, p') \leqslant ۲KC_{k-۱}(S)$$

□

تعریف ۴ *فرض کنید $(P, dist)$ متریک ما باشد. پس برای $\beta \geqslant ۱$ تعریف می‌کنیم: یک مجموعه‌ی هسته‌ی β، برای مجموعه نقاط $S \subset P$ هر زیرمجموعه‌ی $S' \subset S$ باشد که برای هر زیرمجموعه‌ی $F \subset P$ نقطه‌ی $(k-۱)$ داشته باشیم: $KC(S', F) \geqslant \beta KC(S, F)$.*

تعریف ۵ *فرض کنید $(P, dist)$ متریک ما باشد. پس برای هر $\beta \leqslant ۱$ و عدد صحیح l، یک $l - robust\beta - coreset$ برای هر مجموعه نقطه‌ی $S \subset P$ را به این صورت تعریف می‌کنیم: زیرمجموعه‌ی $S' \subset S$ طوری که به ازای هر مجموعه نقاط دورافتاده‌ی $O \subset P$ با حداکثر l نقطه، $S' - O$ یک β-مجموعه‌هسته‌ی $S - O$ باشد.*

۳.۳ هشینگ حساس به محل

هشینگ حساس به محل روشی برای حل مسائل همسایه نزدیک تقریبی است. ایده‌ی اصلی آن هش کردن داده و نقطه کوئری به صورتی است که احتمال برخورد برای نقاطی که نزدیک به هم هستند از آنهایی که از هم دور هستند بسیار بیشتر باشد. تعریف ریاضی آن در ادامه آمده است.

تعریف ۶ *یک خانواده‌ی $H = h : \Delta \to U$ را برای $(\Delta, dist)$، $(r_۱, r_۲, p_۱, p_۲)$-حساس می‌گویند، اگر برای هر $p, q \in \Delta$ داشته باشیم:*

- *اگر $dist(p, q) \leqslant r_۱$ باشد، آنگاه $Pr_H[h(q) = h(p)] \geqslant p_۱$ باشد.*
- *اگر $dist(p, q) \leqslant r_۲$ باشد، آنگاه $Pr_H[h(q) = h(p)] \leqslant p_۲$ باشد.*

برای اینکه یک خانواده‌ی حساس به محل مفید باشد، باید در نامساوی‌های $p_۱ > p_۲$ و $r_۱ < r_۲$ صدق کند.
برای یک خانواده‌ی LSH داده شده، الگوریتم L تابع هش $g_۱, ..., g_L$ و آرایه‌های متناظر آنها یعنی $A_۱, ..., A_L$ را تولید می‌کند. هر تابع هش به شکل $g_i =< h_{i,۱}, ..., h_{i,K} >$ است که $h_{i,j}$ به صورت تصادفی یکنواخت از H انتخاب می‌شود. سپس هر نقطه‌ی p در یک سطل $g_i(p)$ از A_iها ذخیره می‌شود $(۱ \leqslant i \leqslant L)$. برای پاسخگویی به یک کوئری q، ما در $A_۱(g_۱(q)) \cup ... \cup A_L(g_L(q))$ جستجو می‌کنیم. یعنی برای هر آرایه تنها یک سطل که متناظر نقطه کوئری q است را بررسی می‌کنیم.
در این مقاله برای سادگی LSH را برای فاصله‌ی همینگ در نظر می‌گیریم. هرچند نتایج مشابه برای توابع LSH عمومی قابل اثبات هستند. لم زیر را از (۱۰) مرور می‌کنیم.

لم ۳ *فرض کنید $dist(p, q)$ متریک همینگ باشد برای $p, q \in \Sigma^d$ که Σ هر الفبای متناهی باشد. پس برای هر $r, c \geqslant ۱$ وجود دارد خانواده‌ی H که $(r, rc, p_۱, p_۲)$-حساس هستند که $p_۱ = ۱ - r/d$ و $p_۲ = ۱ - rc/d$. همچنین اگر قرار دهیم $\rho = \frac{log ۱/p_۱}{log ۱/p_۲}$ آنگاه داریم $\rho \leqslant ۱/c$ است. به علاوه با اضافه کردن صفرهای قبل از عدد می‌توانیم فرض کنیم $r/d \leqslant ۱/۲$.*

۴ الگوریتم A

الگوریتم A که اولین بار در (۱) ارائه شد، بر مبنای الگوریتم LSH است. در زمان پیش‌پردازش، LSH L تابع هش $g_۱, ..., g_L$ و آرایه‌های $A_۱, ..., A_L$ را می‌سازد. بعد هر نقطه p در سطل‌های $A_i[g_i(p)]$ ذخیره می‌شود (به ازای همه‌ی $i = ۱..L$). به علاوه برای هر آرایه A_i الگوریتم از GMM استفاده می‌کند

تا ۲-تقریبی از k-زیرمجموعه بهینه برای هر سطل پیدا کند و آن را در سطل متناظر A'_i ذخیره می‌کند. این زیرمجموعه یک ۱/۳ هسته برای نقاط سطل است.
یک نقطه کوئری q داده شده است، الگوریتم اجتماع سطل‌ها $Q = A'_1(g_1(q)) \cup ... \cup A'_L(g_L(q))$ را حساب می‌کند و بعد از آن همه‌ی نقاط دورافتاده (یعنی نقاطی که فاصله‌شان تا q بیشتر از cr است) را حذف می‌کند. در گام آخر الگوریتم GMM را روی Q اجرا می‌کند و جواب تقریباً بهینه‌ی k-زیرمجموعه‌ی Q را برمی‌گرداند.
شبه کدها در ادامه آمده‌اند و در بخش بعد به تحلیل درستی الگوریتم می‌پردازیم.

الگوریتم ۲ پیش‌پردازش

```
ورودی: G = {g_1, ..., g_L}: مجموعه‌ی L تابع هش، P: مجموعه نقاط، k
خروجی: A' = {A'_1, ..., A'_L}
به ازای همه نقاط p ∈ P:
    به ازای همه‌ی توابع هش g_i ∈ G:
        نقطه‌ی p را به سطل A_i[g_i(p)] اضافه کن.
به ازای A_i ∈ A:
    به ازای j = 1 → size(A_i):
        کامنت: فقط نقاط تقریب k-متنوع رادر هر سطل نگه دار.
        A'_i[j] = GMM(A_i[j], k)
```

الگوریتم ۳ پردازش کوئری

```
کامنت: ورودی q: نقطه کوئری، k
خروجی Q: مجموعه نقاط k-متنوع
Q ← ∅
به ازای i = 1 → L:
    Q ← Q ∪ A'_i[g_i(q)]
به ازای همه‌ی q ∈ Q:
    اگر dist(p, q) > cr:
        p را از Q حذف کن. کامنت: چون این نقطه دورافتاده است.
Q ← GMM(Q, k)
Q را برگردان.
```

۱.۴ تحلیل

در این قسمت، ابتدا مقادیر پارامترهای L و K را بر حسب n و $\rho \leqslant 1/c$ به دست می‌آوریم، طوری که با احتمال ثابت الگوریتم کار کند. اینجا L تعداد کل توابع هش به کار رفته و K تعداد توابع هش $h_{i,j}$ است که در g_i استفاده شده‌است. همچنین ما باید نشان دهیم محدود کردن اندازه‌ی سطل‌ها به k متنوع‌ترین نقطه در A'، برای به دست آوردن تقریب خوب کارآمد است. این موضوع را در ادامه بحث می‌کنیم.

لم ۴ *به ازای* $c > ۲$، *توابع هش* $g_۱, ..., g_L$ *به شکل* $g_i =< h_{i,۱}, ..., h_{i,K} >$ *وجود دارند که* $h_{i,j} \in H$ *است که* $p_۱$ *و* $p_۲$ *در لم۳ تعریف شده‌اند، طوری که با قرار دادن* $L = (log(۴k)/p_۱)^{۱/(۱-p)}$ × $(۴n)^{\rho/(۱-\rho)}$ *و* $K = \lceil log_{۱/p_۲}(۴nL) \rceil$ *دو پیشامد زیر با احتمال ثابت رخ دهند:*

- $\forall p \in Q* : \exists i, s.t., p \in A_i[g_i(q)]$ *که* $Q*$ *جواب بهینه (*k*-زیرمجموعه بهینه برای* $P \cap B(q, r)$ *است.*
- $\forall p \in \cup_i A_i[g_i(q)] : dist(p, q) \leqslant cr$ *یعنی هیچ داده‌ی دورافتاده‌ای به ازای تمام توابع هش، بین نقاطی که به سطل شامل* q *هش می‌شوند، نباشد.*

اثبات آن در پیوست A *آمده است.*

قضیه‌ی ۱ *چون هر نقطه یک بار در هر تابع هش، هش می‌شود؛ کل فضایی که توسط این الگوریتم استفاده می‌شود حداکثر انقدر است:*

$$nL = n(log(۴k)/p_۱(۴n)^{\rho})^{۱/(۱-\rho)} = O((\frac{nlogk}{۱ - r/d})^{\frac{۱}{۱-\rho}}) = O((nlogk)^{۱+\frac{۱}{c-۱}})$$

که برای به دست آوردن آن از این حقیقت که $c > ۲, \rho \leqslant ۱/c, r/d \leqslant ۱/۲$ *است، استفاده کرده‌ایم. همچنین به* $O(nd)$ *فضا برای ذخیره‌ی نقاط نیاز داریم.*

قضیه‌ی ۲ *زمان کوئری* $O(((logn)/r + k^۲).(logk)^{\frac{c}{c-۱}}.n^{\frac{۱}{c-۱}}d)$ *است.*

اثبات. زمان کوئری الگوریتم برای هر کوئری حداکثر $O(L)$ محاسبه‌ی هش است که هر کدام $O(K)$ زمان می‌گیرد:

$$O(KL) = O((log_{۱/p_۲}(۴nL)).L) = O(\frac{logn}{log(۱/p_۲)}L)$$

$$= O(\frac{d}{r}logn.(\frac{logk}{۱ - r/d})^{\frac{c}{c-۱}}.n^{\frac{۱}{c-۱}})$$

$$= O(\frac{d}{r}(logk)^{\frac{c}{c-۱}}n^{\frac{۱}{c-۱}}logn)$$

که برای به دست آوردن آن از تقریب $logp_۲ \approx ۱ - p_۲ = \frac{cr}{d}, c \geqslant ۲, r/d \leqslant ۱/۲$ استفاده کرده‌ایم. همچنین در قدم آخر، لازم است الگوریتم GMM را حداقل برای متوسط kL [۱] نقطه اجرا کنیم که زمان زیر را لازم دارد:

$$O(k^۲Ld) = O(k^۲.(logk/p_۱(۴n)^{\rho})^{\frac{۱}{(۱-\rho)}}d)$$

$$= O(k^۲(logk)^{\frac{c}{c-۱}}.n^{\frac{۱}{c-۱}}d)$$

□

لم ۵ $GMM(S, k)$ *یک ۱/۳-هسته برای* S *محاسبه می‌کند.*

[۱] $expectation$

اثبات. فرض کنید مجموعه k نقطه محاسبه شده توسط GMM را S' بنامیم. حالا هر زیرمجموعه‌ی $(k-۱)$ عضوی از نقاط $F \subset P$ را بردارید. طبق اصل لانه کبوتری $a, b \in S'$ وجود دارند که نزدیک‌ترین نقطه به آنها در F یکسان باشد؛ یعنی $c \in F$ وجود داشته باشد که:

$$c = argmin_{f \in F} dist(a, f) = argmin_{f \in F} dist(b, f)$$

در نتیجه طبق نامساوی مثلث داریم:

$$div(S') \leqslant dist(a, b) \leqslant dist(a, c) + dist(b, c) \leqslant ۲KC(S', F)$$

حالا هر نقطه‌ی $s \in S$ را بردارید و s' را نزدیک‌ترین نقطه از اعضای S' به s فرض کنید و f را نزدیک‌ترین نقطه از F به s'. همچنین فرض کنید $a \in S'$ نقطه‌ای باشد که در قدم آخر الگوریتم GMM اضافه شده است. پس طبق تعریف s' و a و انتخاب حریصانه‌ی GMM داریم:

$$dist(s, s') \leqslant min_{p \in S' \setminus \{a\}} dist(p, s) \leqslant min_{p \in S' \setminus \{a\}} dist(p, a) \leqslant div(S')$$

بنابرین طبق نامساوی مثلث:

$$dist(s, f) \leqslant dist(s, s') + dist(s', f) \leqslant div(S') + KC(S', F) \leqslant ۳KC(S', F)$$

از آنجایی که این برای هر $s \in S$ برقرار است، می‌توانیم نتیجه بگیریم $KC(S, F) \leqslant ۳KC(S', F)$ که از اینجا حکم نتیجه می‌شود. □

لم ۶ *فرض کنید $S_۱, ..., S_m$ زیرمجموعه‌هایی از P باشند و $S = \cup_i S_i$. همچنین فرض کنید $T_i = GMM(S_i, k)$ یک ۲-تقریب برای k-زیرمجموعه بهینه برای S_i باشد که با اجرای الگوریتم GMM روی S_i به دست آمده است. همچنین تعریف کنید $T = \cup_i T_i$ و فرض کنید $T' = GMM(T, k)$ یک ۲-تقریب برای k-زیرمجموعه بهینه‌ی T باشد. در این صورت داریم: $div(T') \geqslant \frac{۱}{۴} div_k(S)$*

اثبات. فرض کنید S' k-زیرمجموعه بهینه‌ی S باشد. همچنین فرض کنید $a \in T'$ آخرین نقطه‌ای باشد که توسط الگوریتم $GMM(T, k)$ اضافه شده است و قرار دهید $T'- = T' \setminus a$. طبق اصل لانه کبوتری $p, p' \in S'$ و $c \in T'-$ وجود دارند که:

$$c = argmin_{t \in T'-} dist(t, p) = argmin_{t \in T'-} dist(t, p')$$

پس طبق نامساوی مثلث، لم ۵ و این حقیقت که اجتماع β-هسته‌های S_i یک β-هسته برای S است، داریم:

$$\begin{aligned} div_k(S) = div(S') &\leqslant dist(p, p') \leqslant dist(p, c) + dist(p', c) \\ &\leqslant ۲KC(S, T'-) \leqslant ۶KC(T, T'-) \end{aligned}$$

و چون برای هر $b \in T$ داریم:

$$min_{t \in T'-} dist(t, b) \leqslant min_{t \in T'-} dist(t, a) \leqslant div(T')$$

در نتیجه $KC(T, T'-) \leqslant div(T')$ و لم اثبات می‌شود. □

قضیه‌ی ۳ *نتیجه: با احتمال ثابت، ضریب تقریب به دست آمده توسط این الگوریتم ۶ است.*

اثبات. فرض کنید $S_i = A_i[g_i(q)]$ و $S = \cup_i S_i$ باشد. همچنین فرض کنید $T_i = A'_i[g_i(q)]$ و $T = \cup_i T_i$ باشد. به علاوه $Q*$ را k-زیرمجموعه بهینه‌ی $P \cap B(q,r)$ تعریف کنید و $T' = GMM(T,k)$ و Q را مجموعه خروجی فرض کنید. از توضیح الگوریتم، بدیهی است که $Q \subset B(q,cr)$ است. پس تنها چیزی که باید ثابت کنیم تنوع است.
طبق قضیه ۴، با احتمال ثابت دو گزاره‌ی زیر برقرار هستند:

- $S \subset B(q,cr)$ بنابرین داریم $T \subset B(q,cr)$ که نشان می‌دهد وقتی الگوریتم ۳ را روی q اجرا می‌کنیم، چون T شامل هیچ داده دورافتاده‌ای نیست، الگوریتم GMM در خط ۱۰ روی مجموعه T فراخوانی می‌شود و بنابرین $Q = T'$.
- $Q* \subset S$ پس داریم $div_k(S) \geqslant div_k(Q*) = div(Q*)$

پس طبق لم ۶ داریم:

$$div(Q) \geqslant \frac{۱}{۶} div_k(S) \geqslant \frac{۱}{۶} div(Q*)$$

□

۵ الگوریتم B

در این قسمت، نسخه‌ی تغییریافته‌ی الگوریتم A را معرفی و تحلیل می‌کنیم که ضریب تقریب ثابت دارد. فرض کنید می‌دانیم تعداد نقاط دورافتاده در هر سطل حداکثر l است. پس می‌توانیم برای هر سطل آرایه‌ی A، یک $l - robust ۱/۳ - coreset$ در سطل مربوطه در آرایه A' ذخیره کنیم. ابتدا در الگوریتم ۴ نشان می‌دهیم چطور یک $l - robust \beta - coreset$ را پیدا کنیم اگر بدانیم چطور یک $\beta - coreset$ را پیدا کنیم. این الگوریتم (۳) است که مجموعه‌های هسته را پوست می‌کند تا $\beta - corsest$ به دست بیاورد.

الگوریتم ۴ هسته و داده دورافتاده

۱: کامنت: ورودی S: مجموعه نقاط
۲: خروجی S': آرایه‌ای که یک $(l,\beta) - coreset$ برای S است.
۳: $S' \leftarrow \emptyset$
۴: به ازای $i = ۱ \rightarrow (l+۱)$:
۵: $R_i \leftarrow \beta - coreset(S)$
۶: R_i را به انتهای S' اضافه کن.
۷: $S \leftarrow S \setminus R_i$
۸: S' را برگردان.

لم ۷ *فرض کنید $O \subset P$ یک زیرمجموعه از داده‌های دورافتاده باشد و S'_j مجموعه‌ی (kj) نقطه‌ی S' باشد که S' بعد از j-امین مرحله الگوریتم به دست آمده باشد. در این صورت برای هر $l \leqslant j \leqslant ۰$ که در شرط $|Si_{j+۱} \cap O| \leqslant j$ صدق کند، $S'_{j+۱} \setminus O$ یک β-هسته برای $S \setminus O$ است.*

اثبات. فرض کنید $F \subset P$ هر زیرمجموعه‌ای از $(k-1)$ نقطه باشد و q دورترین نقطه از F در بین اعضای $(S \setminus O)$ باشد، یعنی:

$$q = argmax_{p \in S \setminus O} min_{f \in F} dist(p, F)$$

حالا به ازای هر $i \leqslant (j+1)$، اگر q نقطه‌ای در R_i باشد، آنگاه لم برقرار است چون $KC(S, F) = KC(S'_{j+1}, F)$. در غیر این صورت چون q در هیچ یک از $(j+1)$ گام اول انتخاب نشده است، هر کدام از R_i ها (برای $i \leqslant (j+1)$) شامل r_i هستند که $KC(\{r_i\}, F) \geqslant \beta KC(\{q\}, F)$. از بین این $j+1$ r_i، حداقل یکی از آنها در O نیست، پس $KC(S'_{j+1} \setminus O, F) \geqslant \beta KC(S \setminus O, F)$. □

قضیه‌ی ۴ *نتیجه: الگوریتم ۴ به درستی یک (l, β) هسته برای S محاسبه می‌کند.*

اثبات. فرض کنید برای هر مجموعه‌ی $O \subset P$ از نقاط دورافتاده که $|O| \leqslant l$ باشد، شرایط لم ۷ برای $j = l$ برقرار است. پس وقتی الگوریتم جواب را برمی‌گرداند، یک $l - robust\beta - coreset$ را به درستی محاسبه کرده‌است. □

چون طبق لم ۵، می‌دانیم که $GMM(S, k)$ یک ۱/۳-هسته با اندازه‌ی k برای S پیدا می‌کند، کافی است خط ۳ الگوریتم ۴ را با $R \leftarrow GMM(S, k)$ جایگزین کنیم تا به یک $l - robust1/3 - coreset$ با اندازه‌ی $k(l+1)$ برای مجموعه S برسیم.
پس تنها تغییری که در قسمت پیش‌پردازش الگوریتم A انجام می‌دهیم این است که هر سطل از A'_i یک $l - robust1/3 - coreset$ از سطل متناظر A_i را نگه می‌دارد. پس خط ۸ الگوریتم ۲ به این صورت تغییر می‌کند: $A'_i[j] = (l, \beta) - coreset(A_i[j], l = 3L, \beta = 1/3)$.
شبه کد پیش‌پردازش کوئری در الگوریتم ۵ نشان داده شده است. برای هر سطل که متناظر q، الگوریتم سعی می‌کند کوچکترین مقدار l را پیدا کند که تعداد داده‌های دورافتاده در $k(l+1)$ عضو اول $A'_i[g_i(q)]$ از l بیشتر نشود. سپس این نقاط را به T اضافه می‌کند و k زیرمجموعه بهینه تقریبی از نقاط غیر دورافتاده‌ی T را برمی‌گرداند.
توجه کنید که حلقه داخلی (خط ۴ تا ۱۱) الگوریتم ۵ می‌تواند به صورت کارا پیاده‌سازی شود. با دانستن تعداد نقاط دورافتاده‌ی U_i^j، تنها k عضو دیگر هستند که باید چک شوند که در U_i^{j+1} دورافتاده هستند یا نه. همچنین هر نقطه می‌تواند در زمان $O(d)$ چک شود که دورافتاده هست یا نه. پس کل حلقه‌ی داخلی $O(kl_id)$ زمان می‌خواهد.

۱.۵ تحلیل

ابتدا قضیه زیر را که شبیه قضیه ۴ است بیان می‌کنیم. اثبات آن خیلی شبیه اثبات الگوریتم اصلی LSH است که در (۱۰) داده شده است و به همین دلیل حذف شده است.

قضیه‌ی ۵ *توابع هش $g_1, ..., g_L$ به شکل $g_i =< h_{i,1}, ..., h_{i,k} >$ وجود دارند که $h_{i,j} \in H$ که برای H مقادیر p_1 و p_2 طبق قضیه ۳ با قرار دادن $L = log(4k) \times n^\rho / p_1$ و $K = \lceil log_{1/p_2} n \rceil$ به دست می‌آیند. با احتمال ثابت دو رویداد زیر رخ می‌دهند:*

- $\forall pQ* : \exists i, s.t., p \in A_i[g_i(q)]$ که $Q*$ *جواب بهینه است* $(k$*-زیرمجموعه بهینه‌ی* $P \cap B(q, r))$

الگوریتم ۵ پردازش کوئری

۱: کامنت: ورودی: q نقطه ورودی
۲: خروجی: Q: مجموعه‌ی نقاط k-متنوع
۳: $T \leftarrow \emptyset$
۴: مجموعه نقاط دورافتاده $\leftarrow O$
۵: به ازای $L \leftarrow ۱ = i$:
۶: به ازای $۳L \leftarrow ۰ = l$:
۷: $U_i^{l+۱} = A'_i[g_i(q)]$ نقطه‌ی اول از $k(l+۱)$
۸: اگر $|U_i^{l+۱} \cap O| \leqslant l$:
۹: $l_i \leftarrow l$
۱۰: $T_i \leftarrow U_i^{l+۱}$
۱۱: از شرط خارج شو.
۱۲: $T \leftarrow T \cup T_i$
۱۳: $Q \leftarrow GMM(T \setminus O, k)$
۱۴: Q را برگردان.

- $|\{p \in \cup_i A_i[g_i(q)] : dist(p,q) > cr\}| le ۳L$ *یعنی تعداد نقاط دورافتاده از بین نقاط هش شده به سطلی که q به آن هش شده، حداکثر $۳L$ است.*

قضیه‌ی ۶ *نتیجه: چون هر نقطه‌ای در هر تابع هش یک بار هش می‌شود و هر سطل A'_i زیرمجموعه‌ی متناظر سطل A_i است، کل فضای مورد نیاز برای الگوریتم حداکثر انقدر است:*

$$nL = nlog(۴k).n^{\rho}/p_۱ = O(logk.n^{۱+۱/c})$$

دلیل درستی آن این است که $\rho le ۱/c$ است و $p_۱ = ۱ - r/d \geqslant ۱/۲$ است. همچنین ما $O(nd)$ فضا برای نگهداری نقاط نیاز داریم.

قضیه‌ی ۷ *با احتمال ثابت ضریب تقریب الگوریتم ۶ است.*

اثبات. ما ابتدا با مجموعه‌ای از نمادها که برای اثبات لازم است شروع می‌کنیم:

- $S_i = A[g_i(q)], S = \cup_i S_i$
- فرض کنید O مجموعه نقاط دورافتاده باشد، یعنی $O = S \setminus B(q, cr)$. می‌دانیم با احتمال ثابت $|O| \leqslant ۳L$
- فرض کنید S' k-زیرمجموعه بهینه برای $S \setminus O$ باشد.
- $U_i = A'_i[g_i(q)]$ و U_i^j مجموعه‌ی jk عضو اول U_i باشد. (توجه کنید چون خروجی الگوریتم ۴ آرایه است، اعضای آن ترتیب دارند.) $T_i = U_i^{(l_i+۱)}$ تعریف می‌کنیم که l_i طوری انتخاب شده است که به ازای هر $l'_i < l_i$ است و داریم $|U_i^{(l'_i+۱)} \cap O| > l'_i$. این دقیقاً متغیر T_i در الگوریتم ۵ است. به علاوه $T = \cup_i T_i$ قرار دهید.

- $Q*$ را k-زیرمجموعه‌ی بهینه‌ی $P \cap B(q,r)$ تعریف کنید و Q را خروجی الگوریتم یعنی $Q = GMM(T \setminus O, k)$. فرض کنید $a \in Q$ نقطه‌ی اضافه شده در مرحله‌ی آخر الگوریتم GMM باشد. $Q- = Q \setminus \{a\}$ تعریف می‌کنیم.

از توضیح الگوریتم بدیهی است $Q \subset B(q, cr)$. همچنین طبق قضیه ۱، با احتمال ثابت داریم $Q* \subset S$ و تعداد نقاط دورافتاده از $3L$ بیشتر نیست. پس داریم: $div_k(Q) = div(Q*) \leqslant div_k(S \setminus O)$ پس با این شرایط کافی است ثابت کنیم: $div(Q) \geqslant div_k(S \setminus O)/6 = div(S')/6$.
طبق اصل لانه کبوتری، چون $|S'| = k$ و $|Q-| = k-1$ پس نقاط $p, p' \in S'$ وجود دارند که نزدیک‌ترین نقطه به آنها در $Q-$ یکسان است. یعنی $c \in Q-$ وجود دارد که $KC(\{p\}, Q-) = dist(p,c)$ و $KC(\{p'\}, Q-) = dist(p',c)$. پس طبق نامساوی مثلث داریم:

$$div(S') \leqslant dist(p,p') \leqslant dist(p,c) + dist(p',c) \leqslant 2KC(S \setminus O, Q-)$$

طبق لم ۷، $T_i \setminus O$ یک ۱/۳-هسته برای $S_i \setminus O$ است، پس اجتماع آنها $T \setminus O$ یک ۱/۳ هسته برای $S \setminus O$ است؛ پس داریم:

$$KC(S \setminus O, Q-) \leqslant 3KC(T \setminus O, Q-)$$

حالا توجه کنید a در آخرین مرحله‌ی الگوریتم $GMM(T \setminus O, k)$ انتخاب شده است. پس برای هر نقطه $b \in (T \setminus O) \setminus Q$ چون این نقطه توسط GMM انتخاب نشده است، پس b به $Q-$ نزدیک‌تر از a است؛ یعنی باید داشته باشیم: $KC(\{b\}, Q-) \leqslant KC(\{a\}, Q-)$ این یعنی

$$KC(T \setminus O, Q-) \leqslant KC(\{a\}, Q-) \leqslant div(Q)$$

از سه نامساوی قبل حکم نتیجه می‌شود. □

لم ۸ *با احتمال ثابت زمان کوئری* $O((k^2 + \frac{logn}{r})d.logk.n^{1/c})$ *است.*

اثبات. زمان کوئری الگوریتم به ازای هر کوئری سه قسمت دارد. قسمت اول محاسبه $O(L)$ تابع هش است که $O(K)$ زمان می‌برد.

$$O(KL) = O((log_{1/p_2} n).L) = O(\frac{d}{r} logn. \frac{logk}{1 - r/d}.n^{1/c}) = O(logk.n^{1/c}.logn.\frac{d}{r})$$

که برای اثبات آن از تقریب $logp_2 \approx 1 - p_2 = \frac{cr}{d}, c \geqslant 1, r/d \leqslant 1/2$ استفاده شده است. دومین قسمت زمان، مربوط به آخرین گام الگوریتم ۵ است که با احتمال ثابت تعداد کل نقاط دورافتاده حداکثر $3L$ است. پس باید الگوریتم GMM را برای حداکثر $O(kL)$ نقطه اجرا کنیم، یعنی $|T| \leqslant 3L$. پس GMM زمان زیر را می‌برد:

$$O(k^2 Ld) = O(d.k^2 logk.n^{1/c})$$

در نهایت (قسمت سوم زمان الگوریتم)، زمان حلقه داخلی (خطوط ۴ تا ۱۱) الگوریتم ۵، همان طور که قبلاً هم گفته شد می‌تواند طوری پیاده‌سازی شود که کل زمانی که می‌گیرد $O(kl_i d)$ باشد. پس کل زمان اجرای حلقه $O(kd\Sigma_i l_i) = O(kLd)$ است. □

مراجع

[1] G. Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4), 2009.

[2] I. Dinur, S. Safra. The importance of being biased. In *Proc. 34th ACM Sympos. Theory Comput.*, pages 33–42, 2002.

[3] S. Khot, O. Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *Journal of Computer and System Sciences*, 74 (3): 335–349, 2007.