کاوش مجموعه اقلام تکراری جریان‌های داده در مدل پنجره‌ی لغزان حساس به زمان بر مبنای درخت پیشوندی و تخمین احتمالی

محمود دی پیر[1]، حمیدرضا دلیلی اسکویی[2]

۱استادیار، دانشکده رایانه و فناوری اطلاعات، دانشگاه هوایی شهید ستاری، تهران، mdeypir@ssau.ac.ir

۲ استادیار، دانشکده تحصیلات تکمیلی، دانشگاه هوایی شهید ستاری، تهران.

تاریخ دریافت: ۹۴/۱۲/۹ تاریخ پذیرش: ۹۵/۵/۲۰

چکیده

برای کاوش مجموعه اقلام تکراری در جریان‌های داده مدل‌های مختلفی مطرح شده‌اند. مدل پنجره‌ی لغزان حساس به زمان یکی از بهترین این مدل هاست چون به کمک آن هم تغییر مفهوم و هم سرعت متغیر جریان داده ورودی را می‌توان در نظر گرفت. تغییر محتوای پنجره با گذشت زمان، سبب پدیدار شدن الگوهای جدید و حذف برخی از الگوهای قدیمی می‌شود. چگونگی محاسبه یا تخمین تکرار، مجموعه اقلام جدید یکی از عوامل تأثیر گذار در کارایی الگوریتم‌های کاوش الگوهای تکراری در جریان‌های داده است. در این مقاله برای نخستین بار از تخمین احتمالی به منظور تخمین میزان تکرار مجموعه اقلام جدید استفاده شده است. بر اساس این تخمین، الگوریتمی سریع ارائه شده است که قادر است در پنجره‌های حساس به زمان، با میزان حافظه ای قابل قبول، مجموعه اقلام تکراری را کاوش کند. این الگوریتم به منظور ذخیره سازی مجموعه اقلام تکراری پنجره ی فعال از ساختمان داده‌ی جدیدی بر مبنای درخت پیشوندی استفاده می‌کند. آزمایش‌های صورت گرفته بر روی جریان داده‌های واقعی و تولید شده‌ی مصنوعی، نشان دهنده برتری این الگوریتم نسبت به روش‌های ارائه شده قبلی از نظر زمان اجرا و حافظه مصرفی است.

کلیدواژه

جریان کاوی، کاوش مجموعه اقلام تکراری، تخمین احتمالی، پنجره ی لغزان حساس به زمان.

مقدمه

کاوش مجموعه اقلام تکراری، یکی از روش‌های مطرح در داده کاوی است که کاربرد زیادی نیز در علوم مختلف و همچنین در صنعت دارد. ارائه پیشنهادهای ممکن در سیستم‌های توصیه‌گر، تحلیل سبد خرید، تحلیل تصادف وسایل نقلیه، تحلیل پیوند در وب سایت‌های تجاری و تحلیل واژگان در متون مختلف، نمونه‌هایی از این کاربردها هستند. این روش داده کاوی، اولین بار توسط آگروال معرفی شد [۱]، وی الگوریتمی بنام ای. پریوری را نیز برای این مسئله ارائه کرد[۲]. پس از ارائه این الگوریتم این مسئله مورد توجه محققان زیادی قرار گرفت. کاوش مجموعه اقلام تکراری علاوه بر پایگاه‌های داده، در جریان‌های داده نیز مطرح شده است [۳]. یک جریان داده، سریالی از المان‌های داده‌ای است که به صورت پیوسته، نامحدود و با سرعت بالا تولید می‌شود.

با توجه به ویژگی‌های داده کاوی جریانی(یا جریان کاوی)، کاوش مجموعه اقلام در این نوع داده‌ها نیازمندی‌های جدیدی دارد که قبلاً در پایگاه‌های داده مطرح نبود. اولاً هر المان داده باید حداکثر یک بار بررسی شود و این بررسی باید به طور سریع و با در نظر گرفتن محدودیت حافظه موجود انجام شود. علاوه بر این، احتمال تغییر مفهوم در جریان داده با گذشت زمان بسیار زیاد است. تغییر مفهوم در مورد کاوش مجموعه اقلام، به معنی تکراری شدن برخی مجموعه اقلام غیرتکراری و غیرتکراری شدن برخی مجموعه اقلام تکراری در طول کاوش جریان داده ورودی است. برای کاوش جریان داده، مدل پنجره به طور گسترده استفاده می‌شود. بسته به نوع کاربرد سه نوع مدل مختلف پنجره‌ای به نام‌های نشانه[1]، زوال[2] و پنجره‌ی لغزان[3] استفاده می‌شوند. در مدل اول داده‌های بین یک نقطه مشخص از زمان و زمان فعلی مورد توجه‌اند. در مدل دوم بر اساس زمان رسیدن، وزن‌های مختلفی به داده‌ها داده می‌شود. یعنی داده‌هایی که اخیراً رسیده‌اند وزن‌های بیشتری نسبت به داده‌های قبلی می‌گیرند. در مدل سوم تنها طول مشخصی از داده‌های که اخیراً تولید شده، برای عملیات کاوش در نظر گرفته می‌شوند. برای مثال با داشتن پنجره W روی یک پایگاه داده

1 Landmark
2 Damped
3 Sliding Window

تراکنشی تنها آخرین |W| تراکنش و یا همه تراکنش‌ها در |W| واحد زمانی گذشته برای کاوش مورد استفاده قرار می‌گیرند. در اینجا |W| را اندازه پنجره گویند. روش پیشنهادی ما بر اساس مدل سوم یعنی پنجره‌ی لغزان است. با رسیدن تراکنش‌های جدید در این مدل، قدیمی‌ترین تراکنش‌ها منقضی شده و تراکنش‌های جدید به پنجره اضافه می‌شوند که به این عملیات، لغزش پنجره گفته می‌شود. بنابراین پنجره همیشه حاوی جدید ترین تراکنش‌هاست. از آنجا که این پنجره در حافظه اصلی نگهداری می‌شود، اندازه این پنجره محدود است.

مدل پنجره‌ای در تحقیق‌های مختلفی برای یافتن الگوهای تکراری جدید در جریان‌های داده مورد استفاده قرار گرفته است [۱۱-۴]. در این مدل اگر طول پنجره، بر حسب زمان باشد، به آن پنجره حساس به زمان، و اگر طول پنجره بر اساس تعداد تراکنش باشد، به آن پنجره حساس به تعداد تراکنش یا پنجره تراکنشی گفته می‌شود. پنجره‌های حساس به زمان نسبت به پنجره‌های تراکنشی، مدل واقعی‌تری را ارائه می‌دهند چون سرعت متغیر جریان داده ورودی را نیز لحاظ می‌کنند[۱۲،۱۳]. تعداد تراکنش‌های قرار گرفته در یک پنجره زمانی با اندازه زمان مشخص، به سرعت ورود تراکنش‌ها بستگی دارد.

در این مقاله روشی جدید برای کاوش مجموعه اقلام تکراری بر اساس مدل پنجره‌ی لغزان حساس به زمان ارائه می‌شود. علت انتخاب مدل پنجره‌ی لغزان در این مقاله نیاز به حافظه و توان پردازشی کمتر برای کاوش الگوهای تکراری، نسبت به سایر مدل‌هاست. از طرف دیگر در مدل پنجره‌ی لغزان همیشه داده‌های جدید مورد توجه قرار می‌گیرند و از داده‌های قدیمی صرف نظر می‌شود. از آنجایی که داده‌های جدید برای کاربران اهمیت بیشتری دارند، این مدل نسبت به سایر مدل‌ها کاربردی‌تر است. نوآوری‌های ارائه شده در این مقاله عبارتند از: نخست، استفاده از تخمین احتمالی[۴] برای نخستین بار به منظور تخمین تکرار برای مجموعه اقلام تکراری جدید. دوم، استفاده از ساختمان داده‌ای جدید بر مبنای درخت پیشوندی برای نگهداری مجموعه اقلام تکراری پنجره. سوم، ارائه الگوریتمی به منظور کاوش الگوهای تکراری بر مبنای دو مورد اول. ما برای نشان دادن کارایی الگوریتم پیشنهادی خود، آزمایش‌های متنوعی انجام داده‌ایم که نتایج آنها حاکی از کارائی این روش از نظر زمان اجرا و حافظه مصرفی است.

در بخش بعد، روش‌های قبلی کاوش مجموعه اقلام تکراری در مدل پنجره‌ای را مرور خواهیم کرد. سپس در بخش تعریف مسئله، مسئله را به صورت رسمی تعریف می‌کنیم. در بخش الگوریتم پیشنهادی، روش ما به نام پی.تی.اس.ای (PTSA[۵]) معرفی و توصیف می‌شود. در بخش ارزیابی، نتایج مقایسه روش جدید با سایر روش‌های مطرح در این زمینه ارائه می‌شود. این مقاله در بخش نتیجه‌گیری، جمع بندی می گردد.

کارهای انجام شده قبلی

مسئله کاوش الگوهای تکراری در ابتدا در مورد تک قلم‌های تکراری مطرح شد و دو روش بنام های Lossy Counting و Sticky Sampling ارائه شدند[۱۴]. در این روش‌ها فرض بر این است که هر تراکنش شامل یک قلم داده است. به منظور کاوش مجموعه اقلام تکراری در جریان‌های داده نیز روش‌های گوناگونی بر اساس مدل‌های مختلف ارائه شده‌اند. از آنجایی که این مقاله در محدوده کاوش مجموعه اقلام تکراری در مدل پنجره‌ی لغزان است، در این بخش روش‌های مطرح در این زمینه بررسی می‌شوند. در [۸] تعدادی از الگوریتم‌های کاوش الگوهای تکراری، مورد بررسی و مقایسه قرار گرفته‌اند. در [۶] الگوریتمی به نام MFI-TransSW بر اساس الگوریتم ای.پریوری [۲] ارائه شده است که مجموعه کامل مجموعه اقلام تکراری اخیر را در پنجره‌ای از تراکنش‌ها می‌کاود. این الگوریتم از نمایش بیتی برای اقلام موجود در تراکنش‌ها استفاده می‌کند، به طوری که برای هر قلم داده سریالی از بیت‌ها را در نظر می‌گیرد. وجود یک قلم داده در یک تراکنش با بیت ۱ و عدم وجود آن با صفر نشان داده می‌شود. برای اضافه کردن تراکنش جدید و حذف تراکنش قدیمی پنجره، از شیفت به چپ بیتی استفاده می‌کند. این الگوریتم همانند ای.پریوری از تولید و تست مجموعه اقلام کاندید(مجموعه اقلامی که ممکن است تکراری باشند) برای یافتن مجموعه اقلام تکراری بهره می‌برد که این تکنیک کارایی زمانی خوبی ندارد. بنابراین محدودیت ای.پریوری یعنی حجم زیاد الگوهای کاندید در مواقعی که تعداد قلم ها زیاد بوده و مجموعه اقلام تکراری با طول زیاد داریم، در این الگوریتم وجود دارد. از طرفی چون این الگوریتم هم برای وجود تکرار و هم عدم وجود تکرار یک قلم، اطلاعات نگهداری می‌کند، نیازمند حجم زیاد حافظه است. برخی از روش‌های جریان کاوی مجموعه اقلام تکراری از درخت برای ذخیره سازی محتویات پنجره استفاده می‌کنند. این روش‌ها عمدتاً توسعه یافته روش اف.پی.گروت [۱۵] هستند، که یک روش درختی برای کاوش مجموعه اقلام بدون تولید مجموعه اقلام کاندید، در پایگاه داده‌های ایستا است. درخت پیشوندی، ساختمان داده رایج برای کاوش الگوهای تکراری است که در کاوش الگوهای تکراری به صورت افزایشی نیز مورد توجه قرار گرفته است[۱۶]. درخت پیشوندی درختی است که شاخه‌های آن از بالا به پایین مرتب شده هستند و به این ترتیب برای ذخیره‌سازی مجموعه اقلام یا تراکنش‌ها، پیشوند آنها را به اشتراک می‌گذارد و در نتیجه فضای کمتری اشغال می‌شود. این نوع درخت در بسیاری از الگوریتم‌ها و روش‌های کاوش الگوهای تکراری در جریان‌های داده مورد استفاده

4 Probabilistic Estimation
5 Probabilistic based Time Sensitive Algorithm

قرار گرفته است. از جمله این روش‌ها الگوریتم DSTree [۵] است. این الگوریتم تراکنش‌های یک پنجره را به تعدادی دسته تقسیم می‌کند و اطلاعات دسته‌ها را در یک درخت پیشوندی نگهداری می‌کند. نودهای درخت نشان دهنده قلم‌ها هستند که براساس معیار مشخصی مانند ترتیب الفبایی مرتب شده‌اند. اطلاعات تراکنش‌ها به صورت دسته‌ای، به درخت اضافه و حذف می‌شوند. در صورت درخواست کاربر، مجموعه اقلام کل پنجره توسط الگوریتمی شبیه اف.پی.گروت کاوش می‌شوند. از جمله روش‌های موفق درختی دیگر روش CPS-Tree [۷] است. این روش بسیار شبیه DSTree است، با این تفاوت که به طور پویا ساختار درخت جهت کاهش میزان حافظه مصرفی، باز سازی می‌شود تا اقلام همواره به ترتیب نزولی میزان تکرارشان در درخت از بالا به پایین قرار گیرند. بازسازی به موقع درخت و کم کردن حجم آن سبب کاهش حجم حافظه مصرفی و در نتیجه کاوش سریع مجموعه اقلام تکراری می‌شود. از معایب این روش کاهش کارائی به علت بازسازی‌های فراوان درخت در محیط‌های با تغییر مفهوم زیاد است. مظفری و دیگران[۹] الگوریتمی را ارائه داده‌اند که پنجره جریان داده را به تعدادی دسته یا قاب تقسیم می‌کند. در این الگوریتم تنها مجموعه اقلامی برای بررسی مورد توجه قرار می‌گیرند که در یکی از قاب‌ها تکراری باشند. در [۱۰] کاوش الگوهای تکراری در یک افق زمانی محدود مورد توجه قرار گرفته است و الگوریتمی ارائه شده که از یک پنجره‌ی تست به منظور شناسایی و حذف مجموعه اقلام غیرتکراری استفاده می‌کند. در این الگوریتم، مقدار حدآستانه بالای پشتیبانی باعث کاهش اندازه پنجره تست و در نهایت افزایش کارایی این روش می‌شود. شین و دیگران[۱۱] نوع جدیدی از درخت پیشوندی فشرده بنام CP-tree ارائه داده‌اند که می‌توان با کمک آن خلاصه‌ای فشرده از مجموعه اقلام تکراری موجود در یک جریان داده‌ی برخط را نگهداری و به روز رسانی کرد. در این ساختمان داده، هر چه میزان پیمایش و ملاقات یک نود از این درخت بیشتر شود، اندازه درخت کاهش می‌یابد. البته این کاهش اندازه درخت و صرفه جویی در فضای حافظه به قیمت کاهش دقت نتایج، حاصل می‌شود و با آن نسبت مستقیم دارد. نحوه ی پیمایش و به روز رسانی درخت پیشوندی تأثیر زیادی در کارایی الگوریتم ها دارد. در [۱۷] روش جدیدی برای پیمایش درخت پیشوندی ارائه شده است که سبب می‌شود کارایی داده کاوی جریانی مجموعه اقلام به نحو قابل توجهی افزایش یابد.

در کاوش الگوهای تکراری، با توجه به ماهیت جریان داده و پارامترهای ورودی مسئله، ممکن است تعداد بسیار زیادی الگوی تکراری بدست آید که بررسی و به کارگیری آنها را مشکل می‌کند. برای غلبه بر این چالش، در برخی از تحقیقات به جای کاوش تمام الگوهای تکراری، کاوش انواع خاصی از مجموعه اقلام در پنجره‌ی لغزان، مانند مجموعه اقلام بسته تکراری مورد توجه قرار گرفته است[۱۸، ۱۹]. یک مجموعه قلم بسته، مجموعه قلمی است که برای آن نمی‌توان ابر مجموعه‌ای با همان میزان پشتیبانی در پنجره مورد نظر یافت. اندازه ثابت پنجره از دیگر چالش‌های مسئله کاوش الگوهای تکراری در پنجره‌ی لغزان است. در [۲۰] روشی به منظور کاوش الگوهای تکراری در پنجره با اندازه متغیر ارائه شده است. در این روش اندازه‌ی اولیه‌ای توسط کاربر برای پنجره تعیین شده و در ادامه‌ی کار اندازه پنجره با توجه به تغییرات ورودی تنظیم می‌شود.

گاهی کاربران مایلند بجای تعیین اندازه پنجره برحسب تعداد تراکنش، یک مقدار زمانی برای این پارامتر تعیین کنند. به همین خاطر مدل پنجره زمانی مطرح شده است که در این مدل، تراکنش‌هایی که در هر لحظه از جریان داده می‌رسند به عنوان یک دسته یا بلوک در نظر گرفته می‌شوند. با توجه به سرعت متغیر جریان داده ورودی، تعداد تراکنش‌هایی که در هر واحد زمانی از راه می‌رسند، متفاوت بوده و بستگی به سرعت ورود تراکنش‌ها در طول آن واحد زمانی دارد. اولین الگوریتمی که در مدل پنجره زمانی کار می‌کند در [۱۲] ارائه شده است که ما آن را تی.اس.ای(TSA[۶]) نامیم. در این روش، پنجره زمانی به تعدادی بلوک یا دسته تقسیم شده و مجموعه اقلام به طور جداگانه در این دسته‌ها کاوش می‌شوند. لی و دیگران [۱۳] نیز ضمن توصیف بیشتر مدل پنجره‌ی لغزان حساس به زمان، الگوریتمی تقریبی بر اساس این مدل برای کاوش الگوهای تکراری در جریان‌های داده پیشنهاد داده‌اند. این الگوریتم AFIMoTS[۷] نام دارد و از دسته‌بندی مجموعه اقلام بر اساس میزان پشتیبانی آنها استفاده می‌کند. این الگوریتم اگرچه بر اساس مهر زمانی ورود تراکنش‌ها عمل می‌کند ولی به منظور ساده‌تر کردن فرآیند کاوش به حالت تراکنشی ساده تبدیل می‌شود. در این الگوریتم ساختمان داده‌ای درختی طراحی شده است که در آن گره‌ها بر اساس میزان پشتیبانی و همچنین حد آستانه پشتیبانی تعیین شده کاربر، به دسته‌های مختلف تقسیم بندی می‌شوند. البته این دسته بندی‌ها با ورود تراکنش‌های جدید به پنجره و حذف تراکنش‌های قدیمی از آن، به روز رسانی می‌شوند. با توجه به در نظر گرفتن سرعت متغیر داده‌های ورودی در کاربردهای مختلف، ما مدل پنجره‌ی لغزان زمانی را برای تحقیق خود انتخاب کرده‌ایم. لزوم توجه بیشتر به زمان اجرا و حافظه مصرفی با توجه به حجم زیاد داده‌های ورودی و عدم امکان ذخیره سازی دائمی آنها، سبب شد که ما در این مقاله روشی نوین برای جریان کاوی مجموعه اقلام تکراری در پنجره زمانی ارائه دهیم که نسبت به روش‌های ارائه شده‌ی قبلی، زمان اجرای بهتر و حافظه‌ی اصلی مورد نیاز کمتری لازم داشته

6 Time Sensitive Algorithm

7Approximate Frequent Itemset Mining over Time-sensitive Streams

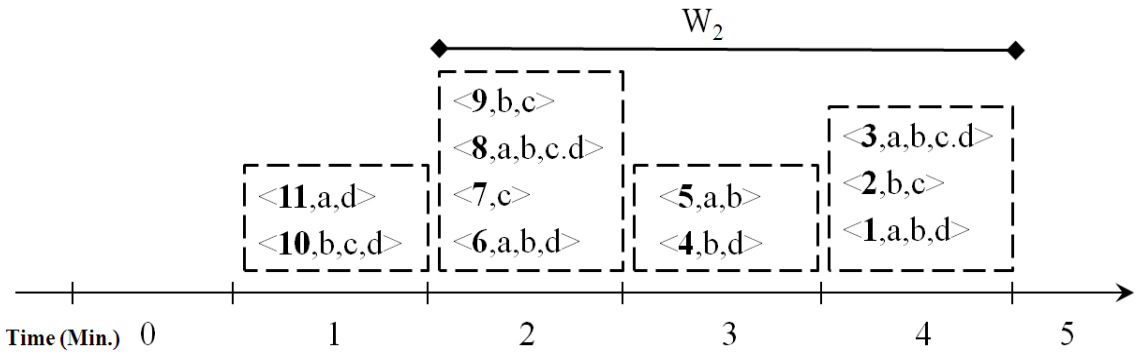
باشد. روش ارائه شده بر مبنای تخمین احتمالی است که بر اساس اطلاعات ما تا کنون در هیچ تحقیقی در حوزه کاوش الگوهای تکراری مورد توجه قرار نگرفته است.

تعریف مسئله

کاوش مجموعه اقلام تکراری در مدل پنجره لغزنده زمانی که اندازه پنجره به صورت مقدار زمانی تعیین می‌شود، کاربردی‌تر است چون این نوع پنجره تغییر سرعت در جریان داده ورودی را در نظر می‌گیرد. از طرف دیگر برای بسیاری از کاربران مشخص کردن اندازه زمانی ساده‌تر و قابل درک‌تر از مشخص کردن اندازه تراکنشی است. به عنوان مثال، به کار بردن مقادیری چون یک ماه گذشته، یک سال گذشته نسبت به یکصد هزار تراکنش گذشته یا یک میلیون تراکنش گذشته، برای کاربران ملموس‌تر است.

فرض کنید $I=\{i_1,i_2,\ldots,i_m\}$ مجموعه‌ای از اقلام باشد. همچنین فرض کنید DS جریانی از تراکنش‌ها به صورت پشت سرهم باشد، به طوریکه هر تراکنش زیر مجموعه ای از I است. برای هر مجموعه قلم (الگو) X که زیر مجموعه ای از I است، گفته می‌شود که تراکنش T در DS شامل الگوی X است اگر $X \subseteq T$ باشد. کسری از تراکنش‌ها در DS که شامل X باشند، به عنوان پشتیبانی X شناخته می‌شود. پشتیبانی مطلق یا تعداد تکرار X، تعداد تراکنش‌هایی در DS است که شامل این مجموعه قلم باشند. با داشتن یک حداقل حد آستانه پشتیبانی S، گوئیم مجموعه قلم X، تکراری است اگر حداقل $S\%$ از تراکنش‌های DS شامل X باشند. پنجره‌ی لغزان روی جریان داده DS شامل $|W|$ تراکنش اخیر در جریان داده DS است، که $|W|$ در اینجا طول پنجره است. طول پنجره در پنجره‌های تراکنشی بر حسب تعداد تراکنش و در پنجره‌های زمانی بر حسب زمان است. با گذشت زمان، پنجره با اضافه شدن تراکنش‌های جدید و حذف تراکنش‌های قدیمی، به طرف جلو حرکت می‌کند. جهت بدست آوردن کارائی بهتر می‌توان تراکنش‌ها را به صورت دسته‌ای اضافه و حذف نمود. برای درک بیشتر پنجره زمانی و پارامترهای آن، مثال زیر را بررسی می‌کنیم.

مثال: شکل ۱، یک پنجره زمانی بر روی یک جریان داده ورودی را نشان می‌دهد. در این شکل، اندازه بلوک و اندازه پنجره به ترتیب یک دقیقه و سه دقیقه تعیین شده‌اند. تراکنش‌هایی که در هر دقیقه از جریان داده رسیده‌اند، در شکل مشخص شده‌اند. به هر دسته از تراکنش‌ها، که در یک زمان دریافت می‌شوند، بلوک گفته می‌شود. با توجه به نرخ متغیر ورود تراکنش‌ها، می‌بینیم که در هر دقیقه تعداد تراکنش‌های متفاوتی رسیده‌اند. اولین بلوک این پنجره مربوط به دقیقه دوم می‌شود. بنابراین شماره پنجره را W_2 نامیده‌ایم. پنجره فعال ما با توجه به تراکنش‌هایی که تا کنون رسیده‌اند، همین پنجره است. این پنجره زمانی، هر دقیقه یک بار به جلو می لغزد. تعداد تراکنش‌های هر پنجره فعال نسبت به پنجره قبلی متفاوت، و وابسته به سرعت رسیدن تراکنش‌ها در دقایق تشکیل دهنده‌ی آن است. هدف ما یافتن و به روز رسانی مجموعه اقلام تکراری در پنجره فعال با هر لغزش پنجره زمانی است. معمولاً مقداری آستانه‌ای تحت عنوان حداقل حد آستانه پشتیبانی یا تکرار توسط کاربر تعیین می‌شود.



شکل(۱): پنجره زمانی در یک جریان داده نمونه

این مقدار، معیاری برای تکراری بودن یا نبودن مجموعه اقلام است. با حرکت پنجره به جلو، تعدادی از مجموعه اقلام غیر تکراری ممکن است تکراری شوند و از طرف دیگر برخی از مجموعه اقلام تکراری ممکن است غیر تکراری شوند. برای کاوش مجموعه اقلام تکراری و به روز رسانی آنها نیازمند الگوریتمی هستیم که در حداقل زمان و با حداقل حافظه این کار را انجام دهد. از طرف دیگر بتواند مقدار تکرار مجموعه اقلام تکراری جدید را بدست آورد و یا با تقریب خوبی تخمین بزند.

روش پیشنهادی

روش پیشنهادی ما از دو بخش اساسی تشکیل شده است. بخش اول شامل الگوریتم کاوش و به روز رسانی مجموعه اقلام به کمک درخت پایش[1] است. بخش دوم، تخمین احتمالی میزان تکرار مجموعه اقلام جدید به کمک قوانین احتمال است. الگوریتم کاوش و به روز رسانی از تخمین احتمالی به منظور تصمیم‌گیری برای اضافه کردن مجموعه قلم جدید به درخت پایش استفاده می‌کند.

الگوریتم پایش و به روز رسانی

در این الگوریتم از درخت پیشوندی برای ذخیره مجموعه اقلام استفاده می‌کنیم چون فضای کمتری نیاز دارد. هر نود حاوی یک قلم است که نماینده مجموعه قلمی است که با دنبال کردن نودها از ریشه تا این نود بدست می‌آید. هر نود در این درخت دارای یک آرایه است که تکرار مجموعه قلم در بلوک‌های پایه‌ای مختلف را نگهداری می‌کند. این آرایه استخراج بلوک پایه‌ای قدیمی در فاز لغزش را بدون نیاز به خود تراکنش‌ها ممکن می‌سازد. در هنگام لغزش، عناصر این آرایه یک خانه به راست شیفت می‌یابند.

با فرا رسیدن بلوک جدید، مجموعه اقلام تکراری موجود در آن کاوش می‌شوند. برای کاوش، از یک نسخه بهینه شده الگوریتم اکلات [۲۱] استفاده می‌شود، به نحوی که هم مجموعه اقلام تکراری را می‌کاود و هم تکرار مجموعه اقلام فعلی را به روز می‌کند. الگوریتم اکلات یکی از معروف‌ترین الگوریتم‌های کاوش الگوهای تکراری است که نسخه بهینه شده‌ی آن نیز در [۲۲] ارائه شده است. در این الگوریتم، هر لیست موجود در یک بلوک پایه‌ای شرطی نشان دهنده رخداد یک مجموعه قلم است و اندازه آن نشان دهنده تکرار آن در بلوک است.

روش کار در شکل (۲) نشان داده شده است. در درخت سمت چپ به هر نود جدولی شکل، بلوک پایه‌ای شرطی گفته می‌شود. درخت سمت راست، درخت پایش نام دارد که مجموعه اقلام تکراری موجود در پنجره فعال را نگهداری می‌کند. با تولید هر بلوک پایه‌ای شرطی، هر کدام از لیست‌ها بررسی می‌شوند. اگر مجموعه قلم مربوط به آن در درخت موجود بود، تکرار آن به روز می‌شود و در تولید بلوک پایه‌ای شرطی بعدی هم استفاده می‌شود. در صورتی که مجموعه قلم در درخت وجود نداشت و تکرار آن بیش از حد آستانه بود، برای اضافه شدن به درخت بررسی می‌شود. در شکل (۲)، اندازه لیست b در بلوک شرطی B^a برابر ۳ است. در نتیجه تکرار این مجموعه قلم در بلوک جدید برابر ۳ است. از آنجایی که نود متناظر با این مجموعه قلم در درخت پایش وجود دارد، مقدار ۳ به تکرارش افزوده می‌شود.

برای بررسی یک مجموعه قلم جدید، تکرار این مجموعه قلم در بلوک‌های پایه‌ای گذشته پنجره با روشی که خواهیم گفت، تخمین زده می‌شود. در صورتی که مجموع تکرار در بلوک پایه‌ای جدید و تکرار تخمینی آن بیش ازحد آستانه باشد، این مجموعه قلم به درخت اضافه می‌شود. در صورتی که در درخت وجود نداشت و تکرار آن در بلوک پایه‌ای بیش از حد آستانه نبود، برای تولید بلوک‌های پایه‌ای شرطی بعدی مورد استفاده قرار نمی‌گیرد.

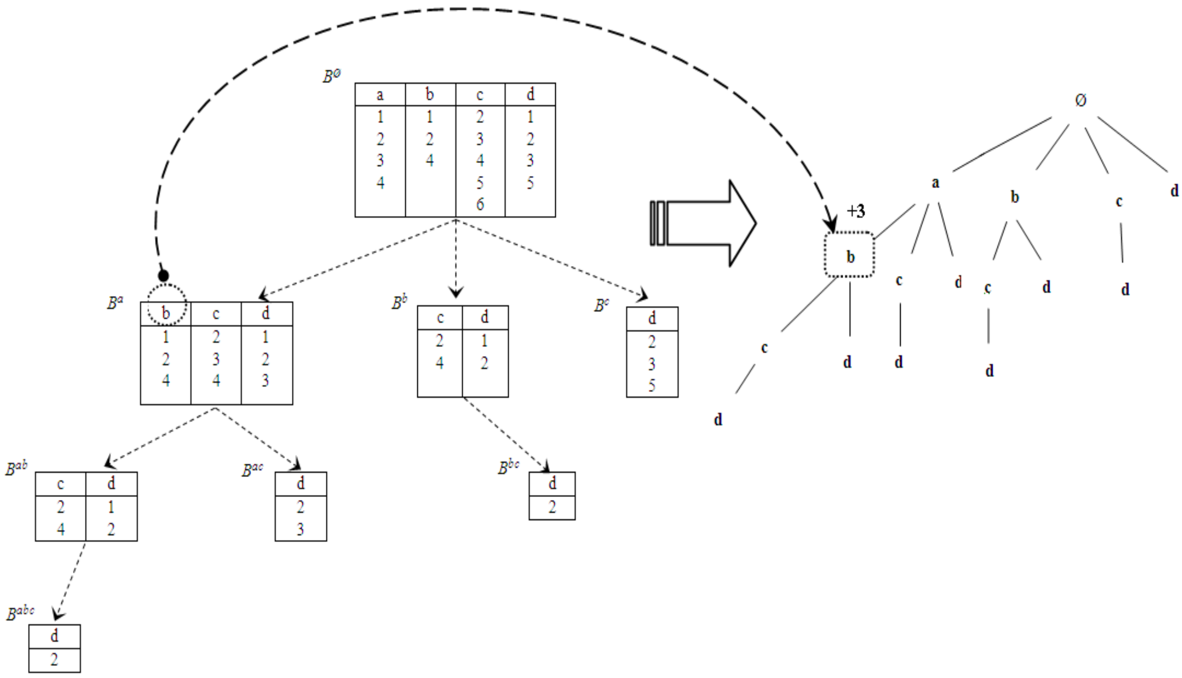
همانطور که ملاحظه شد در این نسخه از الگوریتم اکلات، هم میزان تکرارها به روز می‌شوند و هم مجموعه اقلام جدید کشف می‌شوند. در صورتی که یک لیست، تعداد تکرارش کمتر از حد آستانه باشد ولی مجموعه قلم متناظرش جزء مجموعه اقلام قدیمی باشد، در دور بعدی بازگشت برای تولید بلوک پایه‌ای شرطی جدید مورد استفاده قرار می‌گیرد چون احتمالاً ابر مجموعه‌هایش جزء مجموعه اقلام قدیمی‌اند و می‌بایست میزان تکرار آنها به روز شود.

شبه کد این الگوریتم تحت عنوان کاوش-به روز رسانی بعد از به روز رسانی تک قلم ها در درخت پایش، در شکل (۳) نشان داده شده است. این الگوریتم بر روی بلوک شرطی اقلام، اعمال می‌شود. هر بلوک شرطی تشکیل شده از تعدادی لیست تراکنش‌هاست. در ابتدا تابع با B_i^{ϕ} صدا زده می‌شود که شامل لیست‌های مربوط به تراکنش‌های بلوک جدید است. برای هر لیست تولید شده در خط ۴ در صورتی که مجموعه قلم مربوطه در درخت پایش MT[2] موجود باشد، همانطور که در خطوط ۵ تا ۸ نشان داده شده است، لیست مربوطه برای تولید بلوک شرطی در عمق بعدی مورد استفاده قرار می‌گیرد. علاوه بر این، اندازه لیست به انتهای آرایه Count مجموعه قلم مربوطه اضافه می‌شود. این آرایه، تکرار مجموعه قلم در بلوک‌های پنجره فعال را نگهداری می‌کند. در غیر اینصورت، همانطور که در خطوط ۱۰ تا ۱۸ نشان داده شده است، در صورتی که مجموعه قلم مربوطه در بلوک شرطی تکراری باشد، تکرارش در بلوک‌های گذشته بر اساس روشی که خواهیم گفت، تخمین زده می‌شود. در صورتی که مجموع تکرار تخمینی آن و تکرارش در بلوک جدید بیش از حد آستانه پشتیبانی باشد، این مجموعه قلم به درخت اضافه می‌شود(دستور ۱۳) و لیست تراکنش آن برای تولید بلوک شرطی در عمق پایین‌تر مورد استفاده قرار

1 Monitoring Tree (MT)

2 Monitoring Tree

می‌گیرد(دستور ۱۴). علاوه بر این، همانطور که در دستور ۱۵ نشان داده شده، تکرارش در بلوک جدید به ابتدای آرایه Count افزوده می‌شود. مابقی آرایه، با توزیع تکرار تخمینی در خانه‌های آن پر می شود(خط ۱۶). در دستورات ۲۱ تا ۲۳، در صورتی که بلوک شرطی جدید خالی نباشد، الگوریتم به صورت بازگشتی برای کاوش و به روز رسانی اقلام طولانی‌تر در عمق بعدی، صدا زده می‌شود.



شکل(۲): کاوش و به روز رسانی همزمان مجموعه اقلام تکراری

```
Algorithm: Mine-Update
Inputs: B_i^I ,MT,s
Output: updated MT
1:  For each j ∈ B_i^I do
2:      For each k ∈ B_i^I and  k > j do
3:          tidset(k) = tidset(j) ∩ tidset(k)
4:          e = I ∪ {k}
5:          If e ∈ MT then
6:              B_i^{I∪{j}} = B_i^{I∪{j}} ∪ {k, tidset(k)}
7:              Shift MT[e].Counts
8:              MT[e].Counts[1] = |tidset(k)|
10:         Else if (|tidset(k)| / |B^i|) ≥ s then
11:                 C(e)=Estimate(e,MT)
12:                 If (C(e)+ |tidset(k)|) / |W| ≥ s then
13:                     MT = MT ∪ {e}
14:                     B_i^{I∪{j}} = B_i^{I∪{j}} ∪ {k, tidset(k)}
15:                     MT[e].Counts[1] = |tidset(k)|
16:                     Distribute(C(e), MT[e].Counts[2,...,|W|])
17:                 End if
18:             End if
19:         End if
20:     End for
21:     If B_i^{I∪{j}} ≠ φ then
22:         Mine-Update( B_i^{I∪{j}} ,MT,s)
23:     End if
24: End for
```

شکل (۳): شبه کد الگوریتم اضافه کردن بلوک جدید (کاوش و به روز رسانی)

تخمین احتمالی

همانطور که قبلاً گفته شد، ما نیازمند روشی برای تخمین تکرار یک مجموعه قلم در بلوک‌های قبلی هستیم. یکی از راه‌های تخمین تکرار برای یک مجموعه قلم k تایی، استفاده از تکرار زیر مجموعه‌های k-1 تایی آن و فرمول‌هایی است که در توصیف الگوریتم استوین [۲۳] معرفی شده‌اند. این روش نیازمند بررسی تمام زیر مجموعه‌های k-1 یک مجموعه قلم k تایی است. از طرف دیگر، اگر برای یک مجموعه قلم k تایی، تکرار همه‌ی زیر مجموعه‌های k-1 آن در تمام بلوکهای گذشته پنجره موجود نباشند نمی‌توان تخمینی از تکرار این مجموعه قلم ارائه داد که این خود باعث اضافه شدن دیر هنگام یک مجموعه قلم جدید به درخت پایش می‌شود. البته این روش در جاهایی که از پشتیبانی کاهش یافته استفاده می‌شود، به خوبی عمل می‌کند چون پایش تکرار مجموعه اقلام خیلی زود شروع می‌شود. اما ما برای اینکه کارایی بهتری در مدل پنجره زمانی داشته باشیم از پشتیبانی واقعی استفاده می‌کنیم. ما روشی بر مبنای تخمین احتمالی را به منظور شناسایی مجموعه اقلام تکراری جدید و به عبارت دقیق‌تر برای تخمین تکرار مجموعه اقلام جدید در بلوک‌های قبلی پنجره

ارائه داده‌ایم. اگر ما فرضاً یک مدل دسته‌بندی دو کلاسه داشتیم که برای هر مجموعه قلم جدید، تکراری یا تکراری نبودن را تعیین می‌کرد، این مسئله به سادگی قابل حل بود. اما انجام این کار امکان پذیر نیست، زیرا ما داده های آموزشی برچسب داری در اختیار نداریم که بتوانیم به کمک آن و استفاده از یکی از روش‌های دسته بندی، یک مدل دسته بندی را آموزش دهیم. بنابراین ما راه کار استفاده از تخمین احتمالی را پیشنهاد می‌دهیم. یعنی اینکه تخمین به نحوی باشد که برای یک مجموعه قلم تکراری، احتمالی بالا را بدست آورده و از طرف دیگر بدست آوردن احتمال کم برای یک مجموعه قلم، نشان دهنده‌ی غیرتکراری بودن آن باشد. در واقع میزان پشتیبانی یک مجموعه قلم با احتمال رخداد آن در در تراکنش‌های مختلف پنجره متناسب است. بنابراین ما از تخمین احتمالی برای محاسبه تکرار مجموعه قلم‌های جدید در بلوک‌های قبلی استفاده کرده‌ایم. مجموعه قلم‌هایی که قبلاً در درخت پیشوندی ذخیره شده‌اند نیازی به این تخمین ندارند و فقط مقدار تکرار آنها به روز رسانی می شود.

در روش ما فرض بر این است که یک فرآیند تصادفی پارامتریک با پارامتر a مجموعه اقلام ورودی از جریان داده را تولید می‌کند. پشتیبانی یک مجموعه قلم در یک پنجره را می‌توان به صورت احتمال رخداد آن در یک تراکنش در نظر گرفت. به منظور محاسبه این احتمال، تعداد رخداد این مجموعه قلم در یک پنجره، بر تعداد تراکنش‌های آن پنجره تقسیم می‌شود. به عبارت دیگر این احتمال برابر پشتیبانی این مجموعه قلم است. در الگوریتم ما، اگر در حال اضافه نمودن بلوک(دسته) B_i به پنجره W باشیم و مجموعه قلم X، یک مجموعه قلم تکراری جدید باشد، احتمال رخداد این مجموعه قلم در تراکنش‌های |W|-1 بلوک قبلی برابر است با:

$$P_{W-1}(X)=\frac{C(X)}{\sum_{j=i-|W|+1}^{i-1}|B_j|}=S_{W-1}(X) \qquad (۱)$$

در این فرمول P، C، S، $|B_j|$ و |W| به ترتیب معرف احتمال وجود X در یک تراکنش، تعداد رخداد X، پشتیبانی این مجموعه قلم، اندازه بلوک jام برحسب تعداد تراکنش‌ها و اندازه پنجره بر حسب تعداد بلوک‌های پایه‌ای هستند. با توجه به اینکه شماره آخرین بلوک ورودی i بوده است، شماره بلوک قبلی آن i-1 است. از طرف دیگر با توجه به اینکه تعداد بلوک موجود در پنجره برابر |W| است، شماره اولین بلوک پنجره i-|W|+1 خواهد بود. ما می‌خواهیم در مخرج رابطه (۱) تعداد تراکنش‌های پنجره بدون در نظر گرفتن بلوک iام را بدست آوریم. بنابراین مجموع اندازه بلوک‌های قبلی پنجره یعنی بلوک‌ها با شماره i-|W|+1 تا i-1 را حساب کرده‌ایم. در رابطه (۱) هدف محاسبه احتمال رخداد مجموعه قلم X در بلوک‌های قبلی پنجره است که با تقسیم تعداد رخداد این مجموعه قلم یعنی C(X) بر تعداد کل تراکنش‌های موجود در این بلوک‌ها می‌تواند بدست آید. ما قادر به محاسبه این احتمال نیستیم چون C(X) را در اختیار نداریم. اما با فرض مستقل بودن احتمال رخداد زیر مجموعه‌های این مجموعه قلم، می‌توان احتمال فرمول بالا را بر حسب زیر مجموعه‌های آن تخمین زد. دو زیر مجموعه این مجموعه قلم بنام های Y و Z را در نظر بگیرید به طوری که حاصل اجتماع آنها برابر خود X شود یعنی:

$$\forall X\subseteq I,\exists Y\subset X,\exists Z\subset X\,|\,Y\cup Z=X \qquad (۲)$$

با فرض اینکه Y و Z مستقل هستند، احتمال رخداد مجموعه قلم X با استفاده از احتمال رخداد مجموعه قلم های Y و Z برابر با احتمال اشتراک رخداد این دو مجموعه قلم است. برای یک مجموعه قلم X، این زیر مجموعه‌ها را زیر مجموعه‌های تشکیل دهنده می‌نامیم. به بیان دیگر، در صورتی که اجتماع زیر مجموعه‌هایش برابر X شود ما می توانیم از روی میزان رخداد زیر مجموعه‌هایی از X که با هم تشکیل کل آن را می دهند، تعداد رخداد X را تخمین بزنیم. البته ممکن است میزان رخداد زیر مجموعه‌ها به تنهایی بیش از رخداد X شود. به همین خاطر ما از اشتراک احتمالات رخدادهای زیر مجموعه‌های آن استفاده می‌کنیم تا به تخمین بهتری برسیم. برای یک مجموعه قلم با طول بیش از دو، تعداد زیر مجموعه‌های تشکیل دهنده بیش از دو جفت خواهد بود. فرض کنید متغیرهای تصادفی مربوط به رخدادهای Y و Z در تراکنش‌های یک پنجره، مستقل از هم بوده و به ترتیب با A و B نشان داده شوند. در صورتی که متغیر تصادفی مربوط به رخداد X را نیز D بنامیم، رابطه زیر برقرار خواهد بود:

$$P(D)=P(A\cap B) \qquad (۳)$$

یعنی احتمال رخداد یک مجموعه قلم در یک تراکنش برابر با احتمال رخداد همزمان هر دوی زیر مجموعه‌های تشکیل دهنده‌ی آن است. در نتیجه، پشتیبانی احتمالی X که با PS(X) نشان داده می‌شود را می‌توان به صورت زیر محاسبه کرد:

$$PS(X)=P(D)=P(A\cap B)=P(A)\times P(B)$$
$$=\frac{C(Y)}{\sum_{j=i-|W|+1}^{i-1}|B_j|}\times\frac{C(Z)}{\sum_{j=i-|W|+1}^{i-1}|B_j|}=\frac{C(Y)\times C(Z)}{(\sum_{j=i-|W|+1}^{i-1}|B_j|)^2} \qquad (۴)$$

از آنجایی که ما می‌خواهیم تکرار مجموعه قلم X را در بلوک‌های گذشته پنجره تخمین بزنیم، امید ریاضی متغیر تصادفی D در |W|-1 بلوک گذشته پنجره را به صورت زیر بدست می‌آوریم:

$$PC(X) = Exp(D) = \sum_{j=|W|-i+1}^{i-1}(|B_j| \times P(D)) = P(D) \times \sum_{j=i-|W|+1}^{i-1}|B_j|$$

$$= \frac{C(Y) \times C(Z)}{(\sum_{j=i-|w|+1}^{i-1}|B_j|)^2} \times (\sum_{j=i-|W|+1}^{i-1}|B_j|) = \frac{C(Y) \times C(Z)}{\sum_{j=i-|W|+1}^{i-1}|B_j|}$$

$$PC(X) = \frac{C(Y) \times C(Z)}{\sum_{j=i-|W|+1}^{i-1}|B_j|} \qquad (۵)$$

همانطور که در رابطه ی بالا دیده می شود، احتمال رخداد مجموعه قلم X در تعداد تراکنش بلوک‌های قبلی پنجره ضرب شده است تا میزان شمارش احتمالی[3] این مجموعه قلم در تراکنش‌های قبلی بدست آید. این میزان رخداد احتمالی، همان امید ریاضی تعداد رخداد مجموعه قلم X است. این تکرار تخمینی مربوط به کل بلوک‌های گذشته پنجره است و باید بر حسب اندازه این بلوک‌ها، بینشان تقسیم شود. اگر اندازه هر بلوک i از پنجره W با $|B_i|$ نشان داده شود، تکرار تخمینی مجموعه قلم X در هر بلوک B_k به صورت زیر قابل محاسبه است:

$$PC_k(X) = \left\lceil \frac{|B_k|}{\sum_{j=i-|W|+1}^{i-1}|B_j|} \right\rceil \times PC(X) \qquad (۶)$$

بدیهی است که برای یک مجموعه قلم X در یک بلوک، مقدار بدست آمده از رابطه‌ی بالا ممکن است با مقدار واقعی تکرار در آن بلوک، متفاوت باشد، چون عبارت بالا یک تخمین است که با فرض استقلال پیشامدها محاسبه شده است. این تخمین فقط به ما این اجازه را می‌دهد که این مجموعه قلم را به عنوان یک مجموعه قلم جدید به درخت اضافه کنیم. بدیهی است که با گذشت زمان و فرا رسیدن تراکنش‌های جدید میزان تکرار این مجموعه قلم واقعی‌تر می‌شود و اگر این مجموعه قلم واقعاً تکراری نباشد پس از مدتی از درخت حذف می‌شود. از طرف دیگر اگر این تخمین درست باشد و تراکنش‌های بعدی هم پشتیبانی لازم را فراهم کنند، این مجموعه قلم در درخت پیشوندی باقی خواهد ماند و مقدار تکرارش نیز با گذشت زمان واقعی‌تر خواهد شد.

برای یک مجموعه قلم k تایی، دو زیر مجموعه تشکیل دهنده آن که کمترین پشتیبانی را دارند، بهترین گزینه برای تخمین پشتیبانی آن بر اساس عبارت (۳) هستند چون پشتیبانی آنها نزدیک به پشتیبانی این مجموعه قلم است. اما در الگوریتم ما برای کارایی بهتر و همچنین اضافه شدن زود هنگام مجموعه اقلام به درخت پایش، این شرط در نظر گرفته نشده است. برای یک مجموعه قلم در صورتی که یک جفت زیر مجموعه تشکیل دهنده وجود نداشت، از تعداد بیشتری زیر مجموعه‌های کوچکتر که اجتماعشان برابر آن مجموعه قلم شود، استفاده می‌کنیم. همانطور که گفته شد، در صورتی که مجموع تکرار تخمینی و تکرار در بلوک جدید برای یک مجموعه قلم بیش از حد آستانه شد، این مجموعه قلم به عنوان یک نود جدید به درخت پایش اضافه می‌شود. پس از آن تکرارش در بلوک‌های پایه‌ای بعدی محاسبه شده و در آرایه‌ای بنام Count قرار می‌گیرد. در این آرایه، تکرار این مجموعه قلم در بلوک‌های پایه‌ای ذخیره می‌شود. در یک بلوک ممکن است تکرار واقعی یا تخمینی را داشته باشیم. بعد از اضافه شدن بلوک جدید به پنجره، قدیمی‌ترین بلوک باید حذف شود تا اندازه پنجره از نظر تعداد بلوک‌ها ثابت بماند. عملیات حذف اطلاعات بلوک قدیمی، بوسیله آرایه تکرارهای هر نود درخت انجام می‌گیرد. درخت پایش به منظور استخراج اطلاعات بلوک قدیمی پیمایش می‌شود. برای هر نود، سمت چپ ترین خانه آرایه Count برابر صفر قرار می‌گیرد. علاوه بر این، برای هر نود در صورتی که تکرار کلی‌اش کمتر از حد آستانه شود، این نود و فرزندانش از درخت حذف می‌شوند. علت حذف فرزندان، ویژگی ای. پریوری مجموعه اقلام است که می‌گوید ابر مجموعه یک قلم غیر تکراری، حتماً غیر تکراری است. بنابراین برای حذف تأثیر تراکنش‌های قدیمی پنجره، الگوریتم پیشنهادی نیازی به خود تراکنش‌ها بر خلاف الگوریتم‌های ارائه شده در [۵ و ۷] ندارد. علاوه بر این، نیازی هم به استفاده از جدول کاهش مطابق آنچه در [۱۲] استفاده شده، نیست.

ارزیابی

به منظور ارزیابی کارایی الگوریتم پیشنهادی، آن را با زبان C++ و کتابخانه STL پیاده سازی کرده‌ایم. ما روش خود را با الگوریتم‌های TSA [۱۲] و AFIMoTS [۱۳] مقایسه کرده‌ایم. علت انتخاب این الگوریتم‌ها برای مقایسه این است که هر دو در مدل پنجره‌ی لغزان حساس به زمان عمل کرده و هر دو هم مانند الگوریتم ما تقریبی هستند. برای ارزیابی این روش‌ها از سه مجموعه داده BMS-POS، Kosarak و T40I10D100K استفاده کرده‌ایم. ما هر سه الگوریتم را از دید حافظه مصرفی و زمان اجرا که دو عامل مهم در داده کاوی جریانی است، مقایسه کرده‌ایم. علاوه بر این، درستی نتایج الگوریتم ها با مقایسه خروجی هریک با مجموعه کامل اقلام تکراری، بررسی شده است. همه آزمایش‌ها بر روی کامپیوتر داری پردازنده 2.88GHz و حافظه اصلی ۲ گیگابایت انجام شده‌اند. ما آزمایش‌های خود را با استفاده از مجموعه داده‌های واقعی و مصنوعی که از نظر فشردگی نیز

3 Possible Counting

متنوعند، انجام دادهایم. ویژگیهای این مجموعه دادهها در جدول(۱) نشان داده شدهاند. دو مجموعه داده اول واقعی و سومی مصنوعی است. همانگونه که این جدول نشان میدهد این مجموعه دادهها دارای ویژگیهای متنوعی هستند. ستون آخر این جدول میزان فشردگی هر مجموعه داده را نشان میدهد. این مقدار توسط فرمول زیر برای هر مجموعه داده محاسبه شده است:

(۷) میزان فشردگی = (تعداد قلم داده/متوسط طول تراکنش)*۱۰۰

در مجموعه دادههای فشرده، این مقدار عددی بالا است و در مجموعه دادههای پراکنده این مقدار، عدد کمی است. اگر چه فشردگی مسئلهای نسبی است ولی در برخی تحقیقها [۲۵،۲۴]، اگر میزان فشردگی برای یک مجموعه داده بیش از ۱۰ باشد، به آن مجموعه داده فشرده و در غیر این صورت پراکنده گفته میشود.

جدول(۱): ویژگیهای مجموعه داده های مورد استفاده در آزمایشها

مجموعه داده	تعداد تراکنش	تعداد قلم داده	متوسط طول تراکنش	میزان فشردگی
BMS-POS	۵۱۵۵۹۷	۱۶۵۷	۶،۵۳	۰،۳۹
Kosarak	۹۹۰۰۰۲	۴۱۲۷۰	۸،۱۰	۰،۰۲
T40I10D100K	۱۰۰۰۰۰	۹۴۲	۳۶،۶۱	۴،۲۰

در هر سه الگوریتم، پنجرههای فعال با اضافه کردن تراکنشهای جدید و حذف قدیمیترین تراکنشها از پنجره تشکیل میشوند. این روند تا زمانی که جریان داده تولیدی به کمک یکی از مجموعه دادههای فوق پایان بپذیرد، ادامه مییابد. اما این یک شبیه سازی است و جریان دادههای واقعی بی پایان هستند و انتهایی برای آنها متصور نیست. با این وجود درستی و کارایی هر الگوریتم را میتوان با این شبیهسازی ارزیابی کرد. واحد اضافه و حذف تراکنشها در الگوریتم پیشنهادی بلوک تراکنشهاست ولی در الگوریتم منبع [۱۳] واحد، تراکنش است. به منظور مقایسه عادلانه، در این الگوریتم هم اضافه و حذف شدن را به صورت بلوکی در نظر گرفتهایم ولی در عمل این کار تراکنشی انجام میگیرد.

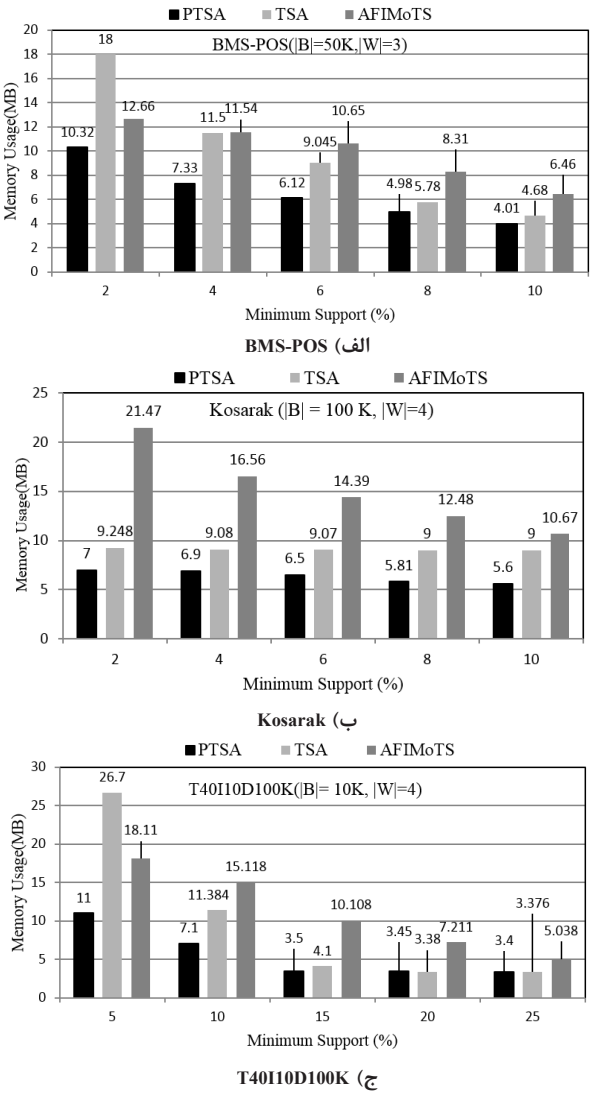
زمان اجرا و حافظه مصرفی دو ویژگی هر الگوریتم جریان کاوی محسوب میشوند. این دو مقدار هر چه کمتر باشند، بهتر است چون جریان داده با سرعت زیاد وارد سیستم میشود و از طرف دیگر برای پردازش دادههای بی پایان و حجیم، هر سیستم کامپیوتری حافظه محدودی دارد. ما زمان اجرا و حافظه مصرفی را برای همه پنجرههای فعال اندازه گرفته و نتایج را میانگینگیری و گزارش کردهایم. از آنجایی که حد آستانه پشتیبانی تأثیر مستقیم روی زمان اجرا دارد، یک محدوده از حدود آستانه مختلف در آزمایش زمان اجرا، مورد استفاده قرار گرفتهاند. از طرف دیگر حافظه مصرفی تحت تأثیر مستقیم اندازه پنجره است. به همین دلیل در آزمایش حافظه مصرفی، این پارامتر را با مقادیر کوچک، بزرگ و متوسط پنجره مقدار دهی کرده و در هر مورد حافظه مصرفی را اندازه گرفتهایم. برای بدست آوردن اندازههای مختلف پنجره، تعداد بلوکهای مختلفی را برای هر اندازه پنجره در نظر گرفتهایم.

برای مدل کردن سرعت متغیر جریان داده ورودی، ما از فرآیند پواسن برای تعیین تعداد تراکنشهایی که در هر واحد زمانی می رسند، استفاده کردهایم. بر این اساس، ما برای هر مجموعه داده از مقدار مشخصی برای میانگین اندازه بلوکهایی که از جریان داده می رسند، استفاده کردهایم. این مقدار را به عنوان مقدار میانگین فرآیند پواسن در نظر گرفتهایم. بنابراین، فرض میشود که در هر واحد زمانی(مثلاً هر دقیقه)، تعداد تراکنشهای ورودی توسط فرآیند پواسن تعیین میشود. سپس ما این تعداد تراکنش را از فایل مجموعه داده میخوانیم و در اختیار الگوریتم قرار میدهیم. به این ترتیب سرعت متغیر جریان داده شبیه سازی می شود.

حال به توصیف نتایج آزمایشهای انجام شده بر روی سه مجموعه داده میپردازیم. شکلهای (۴) و (۵) به ترتیب میزان حافظه مصرفی و زمان اجرای سه الگوریتم را با هم مقایسه میکنند. محور افقی در این شکلها، حداقل حد آستانه پشتیبانی را نشان میدهد و محور عمودی در شکلهای (۴) و (۵) به ترتیب میزان حافظه مصرفی و زمان اجرا را نشان میدهند. میانگین اندازه بلوک در پنجره بر حسب تعداد تراکنشی که در هر زمان وارد میشود و اندازه پنجره بر حسب تعداد بلوک در بالای هر نمودار مشخص شده است. در این شکلها برای مقایسهی دقیقتر، اعداد بدست آمده در هر مورد در بالای ستونها یا کنار نقاط مشخص شدهاند.

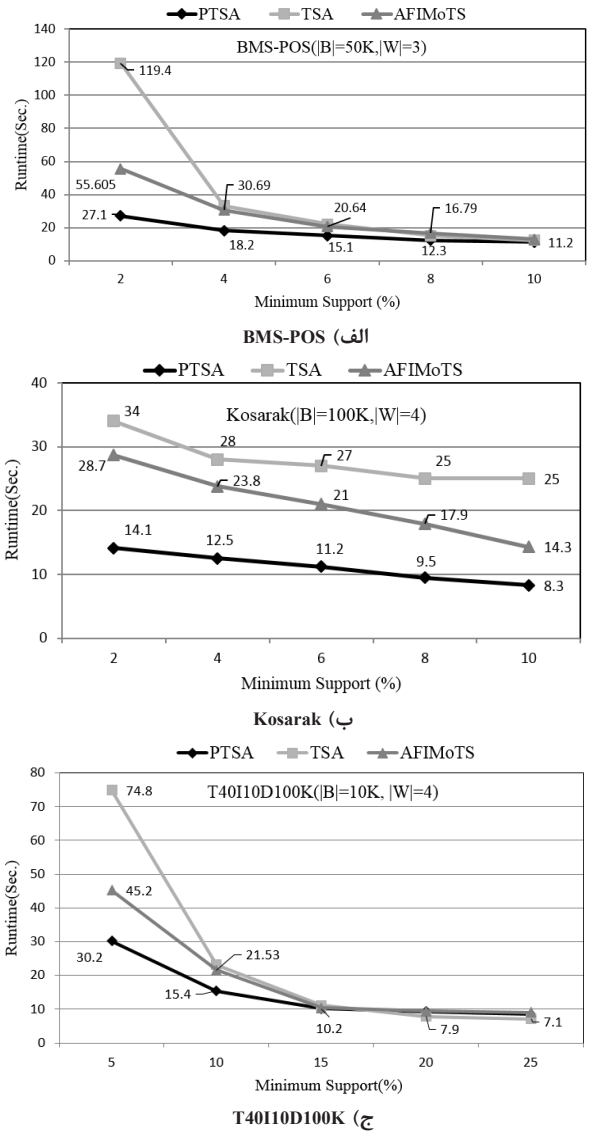
همانطور که در این شکلها دیده میشود، الگوریتم پیشنهادیPTSA، در بیشتر موارد حافظه مصرفی و زمان اجرای بهتری نسبت دو الگوریتم TSA و AFIMoTS دارد. این مسئله در هر سه مجموعه دادهی مورد استفاده، درست است. البته ما بر روی مجموعه داده Kosarak نسبت به دو مجموعه داده BMS-POS و T40I10D100K نتایج نسبی بهتری، هم از نظر حافظه مصرفی و هم زمان اجرا گرفتهایم. در مقادیر کم حدآستانه پشتیبانی، تعداد بیشتری مجموعه اقلام تکراری بدست میآیند زیرا مجموعه اقلامی که رخداد کمی نیز دارند، به عنوان تکراری شناخته میشوند و اینجاست که تفاوت الگوریتمها مشخص میشود. اما هرچه مقدار حدآستانه بیشتر شود تعداد مجموعه اقلام تکراری بدست آمده کمتر میشود و در نتیجه هم حافظه مصرفی الگوریتمها کم میشود و هم زمان اجرا کاهش مییابد. بنابراین تفاوت الگوریتمها

در این حالت چندان زیاد نیست و نمودارها در مقادیر بالای حد آستانه پشتیبانی به هم نزدیک می‌شوند. الگوریتم ما حافظه مصرفی بهتری دارد چون از یک ساختمان داده واحد که همان درخت پایش است، استفاده می‌کند. علاوه بر این، از اشتراک پیشوندی مجموعه اقلام در درخت پایش بهینه نیز بهره می‌برد. از طرف دیگر در الگوریتم TSA از دو جدول مجزا برای نگهداری مجموعه اقلام پنجره و همچنین تکرار مجموعه اقلام در هر بلوک استفاده می‌شود که استفاده از آنها بهینه نیست. در الگوریتم AFIMoTS ساختار پیچیده و مفصلی به منظور نگهدری مجموعه اقلام تکراری و دسته‌بندی‌های آنها استفاده می‌شود که طبیعتاً نیازمند فضای زیادی است.



شکل (۴): مقایسه حافظه مصرفی الگوریتم‌ها برای کاوش مجموعه داده‌های مختلف

شکل (۵) نشان می‌دهد که الگوریتم پیشنهادی زمان اجرای بهتری نیز دارد چون با آمدن هر بلوک پایه‌ای جدید، عملیات کاوش مجموعه اقلام جدید و به روز رسانی تکرار مجموعه اقلام قدیمی در فرآیندی واحد و به طور همزمان انجام می‌گیرند در صورتی که در روش TSA دو عملیات مجزا بر روی هر بلوک جدید برای انجام این کارها صورت می‌پذیرد. علاوه بر این در الگوریتم ما، استخراج بلوک قدیمی در فرآیند لغزش پنجره با پیمایش درخت پیشوندی به سرعت انجام می‌گیرد در صورتی که در الگوریتم TSA این کار با استفاده از جداول کاهش و جدول مجموعه اقلام تکراری صورت می‌پذیرد که بررسی این دو جدول و هماهنگی بین آنها زمانبر است.



شکل(۵): مقایسه زمان اجرای الگوریتم ها برای کاوش مجموعه داده های مختلف

الگوریتم AFIMoTS هم به ازای هر تراکنش ورودی تعیین می‌کند که چه به روز رسانی هایی باید بر روی دسته‌بندی مجموعه اقلام انجام گیرد. از طرف دیگر این کار به ازاء هر تراکنش بلوک وارد شده، انجام می‌گیرد، حال آنکه در روش پیشنهادی به ازاء کل

بلوک وارد شده محاسبات انجام می‌شود. روش تخمین تکرار به کار رفته در الگوریتم پیشنهادی از دیگر دلایل سرعت این روش است زیرا برای محاسبه میزان تکرار یک مجموعه قلم جدید، به محاسبات کمتر و ساده تری نیاز دارد. در الگوریتم پیشنهادی ما، برای استخراج بلوک قدیمی و حذف آن از پنجره زمانی نیازی به پردازش تراکنش‌های مربوطه نیست و این کار با استفاده از اطلاعات ذخیره شده در درخت پیشوندی به سرعت انجام می‌شود. این مسئله یکی از علت‌های برتری زمانی الگوریتم پیشنهادی است.

در آزمایش آخر درستی الگوریتم پیشنهادی را نسبت به سایر الگوریتم‌ها بررسی می‌کنیم. به این منظور میزان خطای مثبت[4] و همچنین خطای منفی[5] این دو الگوریتم در مقادیر مختلف حد آستانه پشتیبانی، با هم مقایسه شده‌اند. منظور از خطای مثبت مجموعه اقلام غیرتکراری هستند که اشتباهاً تکراری شناخته می‌شوند و منظور از خطای منفی مجموعه اقلام تکراری هستند که توسط الگوریتم تشخیص داده نمی شوند. نتایج در جدول ۲ برای مجموعه داده BMS-POS با همان شرایط آزمایش‌های قبلی و مقادیر مختلف حدآستانه پشتیبانی نشان داده شده است. برای سایر مجموعه داده‌ها نیز مقادیر مشابهی بدست می‌آیند. در این جدول، خطای مثبت با FP و خطای منفی با FN نشان داده شده‌اند.

جدول(۲): مقایسه خطای مثبت و خطای منفی الگوریتم‌ها

MinSup(%)	*TSA*		*AFIMoTS*		*PTSA*	
	FP	FN	FP	FN	FP	FN
2	769	616	119	84	121	82
4	150	146	54	29	53	31
6	79	87	26	17	27	14
8	46	35	9	3	7	2
10	21	22	1	0	0	0

همانطور که در این جدول دیده می‌شود الگوریتم ما خطای کمتری نسبت به TSA در مقادیر مختلف حد آستانه پشتیبانی دارد. مقادیر خطای الگوریتم پیشنهادی PTSA مشابه الگوریتم AFIMoTS است و تفاوت‌هایی جزئی با آن دارد. علت داشتن خطای مثبت و منفی کم الگوریتم ما این است که تخمین خوبی را از میزان تکرار گذشته یک مجموعه قلم جدید با استفاده از قوانین احتمال، ارائه می‌دهد. در کل می‌توان گفت که میزان خطای الگوریتم پیشنهادی نسبت به سایر الگوریتم‌ها کم و قابل قبول است.

نتیجه‌گیری

روش تخمین و محاسبه تکرار و نوع ساختمان داده به کار رفته، تأثیر شگرفی در سرعت و حافظه مصرفی برای کاوش مجموعه اقلام تکراری در پنجره‌های زمانی دارد. در این مقاله الگوریتم جدیدی به این منظور ارائه شد که از قوانین احتمالی به منظور تخمین مقدار تکرار مجموعه اقلام تکراری جدید و در نهایت تشخیص آنها استفاده می‌کند. همچنین با بهره‌گیری مناسب از ساختمان داده درخت پیشوندی به منظور ذخیره سازی مجموعه اقلام تکراری در پنجره فعال، زمان اجرا و حافظه مصرفی بهبود می‌یابد. آزمایش‌های صورت گرفته بر روی مجموعه داده‌های واقعی و مصنوعی نشان دهنده‌ی کارایی روش پیشنهادی نسبت به روش‌های ارائه شده قبلی با توجه به پارامترهای مختلف اندازه پنجره، حد آستانه پیشتیبانی و سرعت داده‌های ورودی است. همانطور که اشاره شد الگوریتم پیشنهادی، جوابی تقریبی برای مسئله کاوش الگوهای تکراری در پنجره‌ی لغزان ارائه می‌دهد و در مواردی که جواب دقیق مورد نیاز است کاربردی ندارد.

مرجع‌ها

[1]R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, *Proceeding of the ACM SIGMOD Conference on Management of Data*, 1993, pp. 207–216.

[2]R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proceeding of the VLDB Int. Conf. Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[3]J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, *Data Mining and Knowledge Discovery, 10th Anniversary Issue*, 2007, pp. 55–86.

[4]J.H. Chang, W.S. Lee, Finding recent frequent itemsets adaptively over online data streams, *Proceeding of the ACM SIGKDD Conference*, 2003, pp. 487–492.

[5]C.K.-S. Leung, Q.I. Khan, DSTree: a tree structure for the mining of frequent sets from data streams, *Proceeding of the ICDM, 2006*, pp. 928–932.

[6]H.-F. Li, S.-Y. Lee, Mining frequent itemsets over data streams using efficient window sliding techniques, Expert Systems with Applications 36 (2009) 1466–1477.

[7]S.K. Tanbeer, C. F Ahmed, B.-S Jeong, Y.-K Lee, Sliding window-based frequent pattern mining over data streams, *Information Sciences*, *Volume 179, Issue 22*, *7 November 2009*, *Pages 3843-3865.*

[8]Cheng, J., Ke, Y., & Ng, W. (2008). A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, *16*(1), 1-27.

4 False Positive (FP)
5 False Negative (FN)

[9]B. Mozafari, H. Thakkar, C. Zaniolo, Verifying and mining frequent patterns from large windows over data streams, *Proceeding of the ICDE, 2008* pp. 179–188.
[10]Troiano L, Scibelli G. Mining frequent itemsets in data streams within a time horizon. Data & Knowledge Engineering. 2014, Vol. 89, pp.21-37.
[11]Shin SJ, Lee DS, Lee WS. CP-tree: An adaptive synopsis structure for compressing frequent itemsets over online data streams. Information Sciences. 2014, Vol.278, pp.559-576.
[12]C.-H. Lin, D.-Y. Chiu, Y.-H. Wu, and A.L.P. Chen. Mining frequent itemsets from data streams with a time-sensitive sliding window, Proc. SIAM Int. Conf. on Data Mining, 2005.
[13]H. Li, N. Zhang, J. Zhu, H. Cao, Y. Wang. Efficient frequent itemset mining methods over time-sensitive streams. Knowledge-Based Systems. 56 (2014) 281–298.
[14]G.S.Manku, R.Motwani,Approximate frequency counts over data streams, *Proceeding of the VLDB*, 2002, pp. 346–357.
[15]J. Han, J. Pei, Y. Yin and R. Mao, *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*, Data Mining and Knowledge Discovery, vol. 8, 2004, pp.53-87.
[16]M. Hamedanian, M. Nadimi, M. Naderi. An Efficient Prefix Tree for Incremental Frequent Pattern Mining. International Journal of Information. 2013 Vol. 3(2).
[17]M. Deypir, M.H. Sadreddini, and M. Tarahomi. An Efficient Sliding Window Based Algorithm for Adaptive Frequent Itemset Mining over Data Streams. *J. Inf. Sci. Eng.*, 2013*, Vol. 29*(5), pp. 1001-1020.
[18]Y. Chi, H. Wang, PS. Yu, RR. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on 2004 Nov 1 (pp. 59-66). IEEE.
[19]F.Nori, M. Deypir, and M.H. Sadreddini. A sliding window based algorithm for frequent closed itemset mining over data streams. *Journal of Systems and Software*, 2013, *Vol. 86*(3), pp.615-623.
[20]M.Deypir, M.H. Sadreddini, and S. Hashemi. Towards a variable size sliding window model for frequent itemset mining over data streams. *Computers & industrial engineering*, 2012, Vol. *63*(1), pp. 161-172.
[21]M. Zaki. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, Volume 12, Issue 3:372-390, May/June 2000.
[22]X. Yu, H. Wang. Improvement of Eclat algorithm based on support in frequent itemset mining. Journal of Computers. Vol. 9(9), 2014, pp. 2116-2123.
[23]J.H. Chang, W.S. Lee, estWin: Online data stream mining of recent frequent itemsets by sliding window method, Journal of Information Science, vol. 31, 2005, pp. 76–90.
[24]R.P. Gopalan and Y.G. Sucahy. *High performance frequent patterns extraction using compressed FP-tree*, Proc. The SIAM Int. Workshop on High Performance and Distributed Mining, 2004.
[25]F.-Y. Ye, J.-D. Wang, and B.-L. Shao. *New algorithm for mining frequent itemsets in sparse database*, Proc. Fourth Int. Conf. on Machine Learning and Cybernetics, 2005, pp. 1554–1558.