نشریه بین المللی علوم مهندسی دانشگاه علم و صنعت ایران، شماره ۷، جلد ۱۹، سال ۱۳۸۷، صفحه ۷۴-۶۵
مهندسی شیمی، ریاضی و کامپیوتر

A NEW EFFICIENT METHOD FOR MINING FREQUENT ITEMSETS IN MARKET BASKET DATA ANALYSIS

S.M. Fakhrahmad **M.H. Sadredini** **M. Zolghadri Jahromi**

mfakhrahmad@cse.shirazu.ac.ir **sadredin@shirazu.ac.ir** **zjahromi@shirazu.ac.ir**

Abstract: Discovery of hidden and valuable knowledge from large data warehouses is an important research area and has attracted the attention of many researchers in recent years. Most of Association Rule Mining (ARM) algorithms start by searching for frequent itemsets by scanning the whole database repeatedly and enumerating the occurrences of each candidate itemset. In data mining problems, the size of data is often too large to fit in main memory. However, in some cases such as records of sales of a large supermarket, the probability of a particular item to be present in a transaction is often very low. This is due to the fact that a large number of items are usually available for purchase and also the fact that a small set of items is purchased by a customer in a shopping. In this paper, we make use of these facts to propose an efficient method for mining frequent itemsets. In our approach, the database is scanned just once, and data is encoded into a compressed form and held in a proper data structure in main memory. In each iteration, the time required to measure the frequency of itemsets, is reduced further (i.e., enumerating n-dimensional candidate itemsets is much faster than (n-1)-dimensional itemsets). We evaluate the efficiency of our technique using both synthetic and real-life datasets and compare it with other ARM methods proposed in past research.

روشی کارا برای کاوش مجموعه اقلام پرتکرار در تحلیل داده‌های سبد خرید

سیدمحمد فخراحمد، محمدهادی صدرالدینی و منصور ذوالقدری جهرمی

چکیده: کشف الگوهای پنهان و ارزشمند از درون حجم وسیعی از داده‌های خام، اخیراً توجه بسیاری از محققان را به خود جلب کرده‌است. اغلب روشهای کاوش قوانین تداعی در مرحله اول کار خود کلیه اقلام پرتکرار (ساده و ترکیبی) را از بین تمام اقلام موجود در داده‌ها جستجو می‌کنند که این امر نیازمند به خواندن مکرر کل داده‌ها از دیسک است. در مسائل داده‌کاوی، حجم پایگاه داده‌های تراکنش معمولاً آنقدر زیاد است که قابل بار شدن در حافظه اصلی نمی‌باشند. اما در برخی موارد مانند پایگاه داده‌های تحلیلی مربوط به سبدهای خرید یک فروشگاه، با توجه به تعداد نسبتا زیاد اقلام ممکن (کل اجناس فروشگاه) و نیز محدودیت نسبی اندازه تراکنش‌ها (اقلام خریداری شده در هر سبد)، احتمال رخداد یک قلم داده (خریداری شدن یک کالای خاص) پایین است. در این مقاله با بهره‌گیری از این ویژگی، روشی کارا برای کاوش اقلام پرتکرار در مجموعه داده‌هایی از این قبیل ارائه می‌دهیم. در روش پیشنهادی، داده‌ها تنها یک بار از دیسک خوانده می‌شوند و بعد از آن به یک ساختار رمز شده و خلاصه تبدیل می‌گردند، بطوریکه اولاً قابل نگهداری در حافظه می‌باشند و ثانیاً با توجه به ساختار خاصی که

تاریخ وصول: ۸۵/۱۰/۲۰
تاریخ تصویب: ۸۷/۳/۲۶

س.م. فخراحمد، دانشگاه شیراز، دانشکده مهندسی شماره ۲، بخش مهندسی و علوم کامپیوتر، mfakhrahmad@cse.shirazu.ac.ir
م.ه. صدرالدینی، دانشگاه شیراز، دانشکده مهندسی شماره ۲، بخش مهندسی و علوم کامپیوتر sadredin@shirazu.ac.ir
م. ذوالقدری جهرمی، دانشگاه شیراز، دانشکده مهندسی شماره ۲، بخش مهندسی و علوم کامپیوتر zjahromi@shirazu.ac.ir

دارند، عملیات شمارش به سریع‌ترین نحو ممکن انجام می‌گردد و زمان شمارش دفعات تکرار اقلام در هر مرحله کمتر از مرحله قبل می‌شود. پس از ارائه الگوریتم، کارایی آن را با استفاده از دو مجموعه از داده‌های ساختگی و واقعی ارزیابی کرده و با چند روش کارا که تاکنون ارائه شده‌اند، مقایسه می‌کنیم.

واژه‌های کلیدی: داده‌کاوی، قوانین تداعی، اقلام پرتکرار، تراکنش، تحلیل سبد خرید

۱. مقدمه

کاوش قوانین تداعی یک زمینه تحقیقاتی در مبحث داده‌کاوی[1] و در راستای کشف ارتباطات جالب و با اهمیت بین اقلام اطلاعاتی در بانکهای اطلاعاتی بزرگ و پایگاه داده‌های تراکنشی می‌باشد که اخیراً تلاشهای تحقیقاتی فراوانی را به خود اختصاص داده‌است. از مهمترین کاربردهای آنها می‌توان به مساله معروف تحلیل سبد خرید[2] [۱]، خوشه‌بندی[3] [۲]، رده بندی[4] [۳] و کشف وابستگی‌های تابعی [۴] اشاره کرد. قالب کلی قوانین تداعی بصورت $X \rightarrow Y$ می‌باشد که نشان دهنده نوعی پیشامد همزمان بین اقلام X و Y است (X و Y دو مجموعه قلم دلخواه از پایگاه داده تحلیلی هستند). X و Y به ترتیب مقدم[5] و تالی[6] قانون نامیده می‌شوند. معیارهای مختلفی برای ارزیابی میزان صحت و ارزشمندی قوانین ارائه شده‌اند که بر اساس آنها می‌توان قوانین خوب را از میان مجموعه وسیعی از قوانین ممکن انتخاب کرد. معروفترین و پرکاربردترین این معیارها دو معیار حداقل درجه *پشتیبانی* و حداقل درجه *اطمینان*[7] است. پشتیبانی یک مجموعه اقلام مانند X عبارتست از نسبت تعداد تراکنش‌های شامل تمام اقلام موجود در X به تعداد کل تراکنش‌ها.

در اکثر اوقات، تنها قوانینی برای ما جالب و مفیدند که شامل اقلامی باشند با دفعات تکرار زیاد، نه اقلامی که بندرت در پایگاه داده‌ها یافت می‌شوند. بعنوان مثال، استراتژی چیدمان اجناس یک فروشگاه با دخیل کردن اجناسی که بندرت مشتری دارند، استراتژی موفقی نخواهد بود. بنابراین، اغلب روشها فرضشان بر این است که ما به دنبال مجموعه اقلامی هستیم که حداقل در کسر قابل قبولی از تراکنشها با هم رخ دهند؛ به عبارت دیگر، پشتیبانی آنها از معیار حداقل پشتیبانی مورد نظر ما کمتر نباشد. اصطلاح مجموعه اقلام پرتکرار[8] برای اقلام با پشتیبانی بالا بکار می‌رود. درجه اطمینان یک قانون $X \rightarrow Y$ هم به صورت نسبت تعداد دفعات تکرار همزمان X و Y به تعداد تکرار X به تنهایی تعریف می‌شود، یعنی کسری از تراکنشهای شامل X که Y را نیز شامل می‌شوند. در مجموع، قوانینی مورد قبول واقع می‌شوند که مقادیر قابل قبولی برای هر دو معیار فوق داشته باشند.

در مسائل داده کاوی، حجم داده معمولاً آنقدر زیاد است که قابل بار شدن در حافظه اصلی نمی‌باشد. بنابراین در ارزیابی عملکرد روشهای مختلف کاوش قوانین، معیارهایی از قبیل زمان مورد نیاز برای خواندن داده‌ها از دیسک و یا تعداد دفعاتی که هر جزء داده باید خوانده شود، بکار می‌روند [۵-۷]. تاکنون روشهای مختلفی برای بهتر کردن الگوریتم *Apriori* جهت کشف اقلام پرتکرار توسط محققین مختلف ارائه شده‌است که تلاش اکثر آنها بر این متمرکز است که برای یافتن اقلام پرتکرار تا حد ممکن به‌جای شمارش دفعات تکرار که نیازمند رجوع به دیسک و پایگاه داده‌های حجیم است، به روش مستقیمی بتوان پشتیبانی اقلام را بدست آورد. اگرچه روشهای کارآمد متعددی در این راستا پیشنهاد شده‌اند، اما مساله مهم دیگری کمتر مورد توجه واقع شده‌است و آن اینکه این روش‌ها در موارد خاص و در ارتباط با اقلام بخصوصی جوابگو هستند و در بسیاری از موارد جهت محاسبه پشتیبانی یک مجموعه اقلام راهی بجز شمردن دفعات تکرار وجود ندارد. در این مقاله، سعی کرده‌ایم روشی کارآمد برای کشف قوانین تداعی ارائه دهیم که تکیه اصلی آن بر بهینه سازی عمل شمارش دفعات تکرار مجموعه اقلام داده است.

ادامه مقاله بصورت زیر ترتیب یافته‌است: در بخش ۲، چند مورد از روش‌های مرتبط را که تاکنون ارائه شده‌اند، به اختصار شرح می‌دهیم. در بخش ۳، به یک توصیف کلی از الگوریتم خود به نام *FastFPM*[9] می‌پردازیم. نتایج عملی مقایسه الگوریتم پیشنهادی و تعدادی از الگوریتمهای موجود، در بخش ۴ ارائه و بحث می‌شوند. در پایان نتیجه گیری کلی مقاله را در بخش ۵ ارائه می‌دهیم.

۲. تحقیقات مرتبط

الگوریتم‌های زیادی برای کشف قوانین تداعی تاکنون ارائه شده‌اند. بخش عمده و نسبتا زمانگیر در اکثر الگوریتم‌های موجود از جمله روش پایه ای و معروف *Apriori* [۸] جستجوی اقلام پرتکرار است. برای بهینه سازی این فرایند الگوریتم‌های پیشنهاد شده رویکردهای

[1] *Data Mining*
[2]. *Market basket data analysis*
[3]. *Clustering*
[4]. *Classification*
[5]. *Antecedent*
[6]. *Consequent*
[7]. *Confidence*
[1].*Frequent itemsets*

[2]. *Fast Frequent Pattern Miner*

متفاوتی دارند. تلاش بسیاری از روش‌ها بر کاهش تعداد دفعات مراجعه به دیسک جهت خواندن داده‌هاست.

برای این منظور، بعضی روش‌ها با یافتن راه‌های مستقیم جهت به دست آوردن پشتیبانی بعضی از اقلام، از مراجعات بیهوده به دیسک خودداری می‌کنند و برخی با ساختن ساختمان داده‌های خاصی در حافظه اصلی تا حدی توانسته‌اند به این منظور دست یابند. از کاراترین روش‌های موجود می‌توان از روش‌های *VIPER*[۹]، *ARMOR*[۱۰]، *FP-Growth*[۱۱] و *Closet*[۱۲] نام برد.

VIPER روشی است که از جهت نحوه نمایش تراکنش‌ها شبیه به روش پیشنهادی است. اما به دلیل مراجعات نسبتا زیاد به دیسک کارایی‌اش نسبت به چند مورد از روش‌های دیگر پایین‌تر است. *ARMOR* نسخه بهبود یافته الگوریتم دیگری به نام *Oracle* است که از ساختمان داده‌های بخصوصی به نام DAG استفاده می‌کند و تاکیدش بر بهینه کردن عمل شمارش دفعات تکرار است. برخلاف روش‌های *VIPER* و *ARMOR* که الگوی عملکردشان کاملا بر پایه *Apriori* است، روش‌های *FP-Growth* و *Closet* در فرایند جستجوی اقلام پرتکرار هیچ قلم کاندیدی تولید نمی‌کند. در این الگوریتم در مجموع داده‌ها ۳ بار پیمایش می‌شوند و بعد از آن ساختمان داده خاصی از جنس درخت درهم‌ساز[1] در حافظه ساخته می‌شود که تمام اقلام پرتکرار را می‌توان مستقیما با پیمایش‌های خاصی بر روی این درخت بدست آورد، بدون این که نیازی به تولید اقلام کاندید باشد. اشکال عمده این روش نیاز به حافظه زیاد در ارتباط با مجموعه داده‌های بسیار حجیم است که عملا در بعضی مواقع اجرای الگوریتم را غیر عملی می‌سازد.

روش‌های متعدد دیگری نیز وجود دارند که هریک با دیدگاهی نسبتا متفاوت به کاوش اقلام پرتکرار می‌پردازند [۱۷-۱۳]. بعد از کشف کلیه اقلام پرتکرار از مجموعه داده‌ها، گام بعد یعنی تولید قانون به روشی مستقیم و سریع صورت می‌گیرد. بنابراین روش‌های مختلفی که ارائه می‌شوند، در واقع تفاوتشان در نحوه کشف اقلام پرتکرار است.

روش‌های نامبرده شده تماماً از روش جستجوی پایین به بالا استفاده می‌کنند. بدین معنی که لیست مجموعه اقلام پرتکرار کشف شده در ابتدا خالی است و هر قلم داده‌ای که بعنوان پرتکرار شناخته شود، به لیست اضافه می‌گردد. سپس در هر مرحله ترکیبات مختلف از اقلام پرتکرار مراحل قبل مورد بررسی قرار می‌گیرند. این عمل تا جایی ادامه می‌یابد که بزرگترین ترکیب پرتکرار از اقلام شناخته شود. در کنار این روش‌ها، روش‌های معدودی نیز وجود دارند که از روش جستجوی بالا به پایین استفاده می‌کنند. بدین صورت که کار خود را با یک قلم ترکیبی شامل تمامی اقلام داده آغاز کرده و پرتکرار بودن این ترکیب را بررسی می‌کنند. اگر پرتکرار نبود، هر بار با حذف یک قلم داده از ترکیب، درجه پشتیبانی ترکیب حاصل را محاسبه کرده تا در نهایت بزرگ‌ترین ترکیب‌های پرتکرار پیدا شوند [۲۲-۱۸]. از آنجا که در مسائل واقعی معمولاً تنوع اقلام داده زیاد بوده و از طرفی داده‌ها عموما از طبیعت خلوت برخوردارند، روش دوم (جستجوی بالا به پایین) بشرحی که ذکر شد، در مقایسه با روش اول (جستجوی پایین به بالا) کارایی خوبی ندارد. در مرجع [۲۳] یک مقایسه کلی بین کارایی الگوریتم‌های مختلف بر روی داده‌های واقعی ارائه شده‌است.

در مجموع مشکلات عمده روش‌های ذکر شده، نیاز به دسترسی-های متعدد به دیسک و همچنین شمارش‌های زمانبر اقلام داده جهت کشف مجموعه اقلام پرتکرار است. هدف اصلی از الگوریتمی که در این مقاله ارائه می‌شود، غلبه بر این دو مشکل است.

۳. الگوریتم پیشنهادی

برای قابل فهم‌تر شدن الگوریتم پی‌شنهادی (FastFPM)، از این به بعد پایگاه داده تراکنش‌ها را که دارای هیچ ساختار خاصی نیست، به صورت یک بانک اطلاعات رابطه‌ای با مقادیر دودویی تصور می-کنیم. هر ستون این رابطه نشان دهنده یک قلم داده از کل مجموعه اقلام داده‌های ممکن و هر رکورد متناظر با یک تراکنش از پایگاه داده تراکنشی است. هر مقدار ۱ یا ۰ برای یک قلم داده در یک تراکنش بترتیب نشان دهنده وجود یا عدم وجود آن قلم داده در تراکنش است.

Milk, Bread, Butter
Milk, Butter
Bread, Milk, Beer
Bread, Beer

(الف)

Milk	*Bread*	*Butter*	*Beer*
1	1	1	0
1	0	1	0
1	1	0	1
0	1	0	1

(ب)

شکل۱. دو قالب نمایش یک پایگاه داده تحلیلی مربوط به مساله سبد خرید، (الف) نمایش بدون ساختار تراکنش‌ها، (ب) نمایش ساخت یافته تراکنش‌ها

به عنوان مثال، رابطه نشان داده شده در شکل ۱.(ب) نمایش ساخت یافته پایگاه داده تراکنشی نشان داده شده در شکل ۱.(الف) است که مشتمل بر چهار تراکنش می‌باشد. در مجموعه داده‌های واقعی مربوط به مساله سبد خرید و مسائل مشابه آن تعداد بیت‌های یک به نسبت تعداد صفرها بسیار کم است. دلیل این خاصیت را می‌توان در تعدد اقلام مختلف یک فروشگاه و همچنین محدود بودن ظرفیت هر سبد خرید دانست. بنابراین هرچه یک فروشگاه بزرگتر و تنوع محصولات آن بیشتر باشد، احتمال بیت ۱ در هر ستون از

[3] *Hash Tree*

رابطه شکل ۱.(الف) کمتر می‌شود. شرط اصلی برای کارایی الگوریتم پیشنهادی، وجود این خاصیت در مجموعه داده‌ها است.

در گام نخست از الگوریتم، داده‌ها را بصورت افقی به تعدادی قطعه با طول مساوی (k رکورد در هر قطعه) و بدون اشتراک تقسیم می‌کنیم. در مورد مقدار مناسب برای k، بعدا بحث می‌کنیم. از آنجا که هر ستون از رابطه در هر قطعه خود شامل k بیت است، به مجموعه این k بیت می‌توان به صورت یک کد دودویی k بیتی نگریست که در واقع معادل یک عدد صحیح با دامنه مقادیر بین ۰ و $۲^k - ۱$ می‌باشد. این اعداد صحیح دهدهی عملوندهای اصلی الگوریتم ما هستند.

	A	*B*	*C*
I	1	0	0
	0	1	0
	0	1	0
	1	1	0
II	0	0	1
	1	0	0
	1	0	0
	1	0	0
III	1	0	1
	1	0	1
	1	0	0
	0	0	1
IV	0	1	1
	0	1	0
	0	0	1
	0	1	0
V	1	0	0
	0	0	1
	1	0	0
	1	0	0
VI	0	1	1
	0	1	0
	0	1	0
	0	1	0

شکل۲. مجموعه داده r مشتمل بر ۲۴ تراکنش

رابطه قطعه‌بندی شده فقط یک بار پیمایش می‌شود و پشتیبانی کلیه اقلام تکی (۱ بعدی) محاسبه می‌گردد تا اقلام تکی پرتکرار کشف شوند. در ضمن این عمل، مقادیر دهدهی غیر از صفر (اعداد صحیح معادل بیتهای قطعه‌ها) از تمام قطعه‌ها استخراج می‌شوند. بازاء هر ستون از رابطه در صورتی که قلم داده معادل آن پرتکرار باشد، یک جدول دسترسی مستقیم (جدول درهم ساز[1]) در حافظه می‌سازیم. مقادیر ذخیره شده در این جدول، همان اعداد صحیح غیر صفر هستند که از قطعه‌های مختلف استخراج شده‌اند. کلید دسترسی مستقیم به هریک از مقادیر هم شماره قطعه مربوط به آن می‌باشد (حداقل ۱ و حداکثر برابر با تعداد قطعه‌ها). از آنجا که ما مقادیر صحیح صفر (مربوط به قطعه‌های تماماً صفر) را در جدولهای در هم ساز درج نمی‌کنیم، کلیدهای دسترسی موجود در هر جدول در هم ساز به مناطقی از پایگاه داده‌ها که قلم مربوطه در آن حوالی حداقل یک بار رخ داده، اشاره می‌کنند. اثر مثبت این امر در مراحل بعدی الگوریتم، یعنی جستجوی اقلام با ابعاد بالاتر به مراتب محسوس‌تر است. به این دلیل که در هر مرحله با دور ریختن تعدادی از قطعه‌های صفر فضای جستجو کوچک‌تر می‌شود.

به عنوان مثال، فرض کنید A و B و C سه قلم داده دلخواه باشند و r یک مجموعه داده که وجود یا عدم وجود این اقلام را در ۲۴ تراکنش مختلف نشان می‌دهد (شکل۲) . در این مثال k برابر با ۴ در نظر گرفته شده‌است. اگر حداقل پشتیبانی قابل قبول را برابر با ۰/۴ انتخاب کنیم، با استفاده از الگوریتم پیشنهادی، کل اقلام پرتکرار بصورت زیر جستجو می‌شوند. در گام اول بایستی دفعات تکرار هریک از اقلام تکی شمرده شود و برای هرکدام که پشتیبانی-اش حداقل با حد آستانه برابر باشد، یک جدول در هم ساز به روش مذکور در حافظه ایجاد شود. مقادیر بدست آمده برای پشتیبانی A، B و C به ترتیب برابر ۰/۴۵ ، ۰/۴۱ و ۰/۳۳ است.

از آنجا که پشتیبانی عنصر C از حد آستانه پایین‌تر است، تنها A و B بعنوان اقلام تکی پرتکرار پذیرفته می‌شوند؛ جدول‌های درهم ساز مربوط به اقلام A و B در شکل۳ نشان داده شده است.

I	II	III	V
9	7	14	11

کلیدها
مقادیر

(الف)

I	IV	VI
7	13	15

کلیدها
مقادیر

(ب)

شکل۳. جدول‌های درهم ساز مربوط به (الف) A، (ب) B

جهت محاسبه پشتیبانی یک ترکیب از اقلام مانند AB دیگر نیازی به پیمایش مجدد کل داده‌ها نیست و این کار با استفاده از جدولهای در هم ساز عناصر آن (یعنی A و B) انجام می‌شود. برای این منظور از جدول در هم ساز کوچکتر (شامل عناصر کمتر) شروع می‌کنیم. بازاء هر کلید دسترسی موجود در این جدول، در صورتیکه کلید در جدول دوم هم موجود باشد، دو مقدار صحیح مرتبط با آن از دو جدول در هم ساز را با هم *AND* می‌کنیم. نتیجه عمل *AND* عدد صحیح دیگری است که معادل آن در مبنای ۲ نشان دهنده محل و تعداد رخدادهای همزمان عناصر A و B در قطعه مربوطه است.

صفر بودن عدد حاصل از *AND* نشان‌دهنده این است که در هیچیک از تراکنش‌های آن قطعه وقوع همزمان A و B را نداریم. برای ترکیب AB یک جدول در هم ساز مشابه قبل می‌سازیم و اعداد بدست آمده از عملیات *AND* را به همراه کلیدهای آنها که همان شماره قطعه‌ها هستند، در آن درج می‌کنیم. در این مرحله نیز اعداد صفر را وارد جدول نمی‌کنیم و لذا اندازه این جدول حداکثر

[1]. *Hash Table*

مساوی با اندازه جدول کوچکتر (جدول A یا B) است که البته بندرت اتفاق می‌افتد و معمولاً از هردو جدول اولیه کوچکتر می‌شود. همانطور که ذکر شد، هریک از اعداد صحیح ذخیره شده در جدول AB نماینده رخدادهای همزمان A و B در یک قطعه از داده‌ها است. بنابراین جهت شمارش تعداد این رخدادها کافی است تعداد بیتهای ۱ موجود در نمایش دودویی کلیه اعداد شمرده شوند. ما این کار را از طریق عملیات انتقال به راست (*SHR*)[1] یا انتقال به چپ (*SHL*)[2] انجام می‌دهیم. به این ترتیب که عمل انتقال را روی هریک از اعداد صحیح بطور پیاپی اعمال میکنیم تا زمانیکه مقدار عدد برابر صفر شود. حاصل جمع بیتهای نقلی[3] در مراحل انتقال نشان دهنده تعداد بیت‌های ۱ موجود در آن عدد است. توقف عمل انتقال در زمان صفر شدن عدد، سبب جلوگیری از شمارش‌های بیهوده می‌شود.

با توجه به اقلام تکی پرتکرار بدست آمده در مثال فوق، تنها کاندید ممکن برای مجموعه اقلام ۲ تایی پرتکرار مجموعه *AB* است. برای ساختن جدول در هم ساز برای *AB*، هریک از مقادیر ذخیره شده در جدول در هم ساز عنصر *B* (جدول در هم ساز کوچکتر) بترتیب انتخاب می‌شود تا با مقداری از جدول در هم ساز مربوط به *A* که کلیدشان یکسان است (در صورت وجود)، *AND* شود. تنها کلیدی که در هردو جدول مشترک است، کلید I است. در نتیجه حاصل ترکیب دو جدول، جدولی شامل تنها یک عنصر است. جدول درهم ساز مربوط به قلم ترکیبی *AB* در شکل۴ نشان داده شده است.

کلیدها	I
مقادیر	1

شکل۴. جدول درهم ساز مربوط به AB

برای محاسبه پشتیبانی *AB*، تعداد بیت‌های ۱ تنها عدد صحیح موجود در جدول آن (در مبنای ۲) بایستی شمرده شود. از آنجا که این مقدار برابر با ۰۰۰۱ است، تنها یک عمل انتقال به راست برای شمارش بیت‌های ۱ کافی است. بنابراین در اینجا تعداد مقایسه‌های لازم از ۴۸ مقایسه (۲×۲۴) در دیسک به ۱ مقایسه در حافظه کاهش یافت. اما بطور کلی، این کارایی نسبی در مراحل بالاتر، یعنی جستجو برای مجموعه اقلام پرتکرار با ابعاد بالاتر نمایان‌تر است.

برای بهینه‌تر کردن عمل شمارش، در مواردی که مقدار یک عدد صحیح کوچکتر از $۲^{k/۲}$ باشد، حتما از انتقال به راست استفاده می‌کنیم. دلیل این امر آنست که تمام بیت‌ها در نیمه چپ چنین اعدادی صفر هستند و شمارش آنها که در صورت انتخاب انتقال به چپ اتفاق می‌افتد، لزومی ندارد. به عبارت دیگر با انتخاب عمل انتقال به راست زودتر مقدار عدد به صفر می‌رسد و شمارش کمتری نیاز است. از طرف دیگر، زمانی که مقدار عددی بزرگتر از $۲^{k/۲}$ باشد، بدین معنی است که حداقل یک بیت ۱ در نیمه سمت چپ کد دودویی آن وجود دارد. با توجه به طبیعت خلوت داده‌هایی نظیر مجموعه داده‌های سبد خرید، احتمال اینکه بیت ۱ دیگری در آن قطعه نباشد، بسیار بالاست و در نتیجه استفاده از انتقال به چپ جهت شمارش بیت‌های ۱ ارجحیت دارد.

کارایی ساختار الگوریتم ما در هنگام محاسبه پشتیبانی عناصر با ابعاد بزرگ‌تر محسوس‌تر می‌شود؛ چرا که در هر مرحله اندازه جدولهای در هم ساز ما نسبت به مرحله قبل کوچک‌تر می‌شود و عملیات شمارش سریع‌تر صورت می‌گیرد که این امر هم بدلیل پیدایش اعداد صفر در نتیجه عملیات *AND* می‌باشد که در جدول در هم ساز حاصل درج نمی‌شوند.

در گام بعد، جدولهای در هم ساز مجموعه اقلام ۲ تایی پرتکرار به طریقی مشابه برای یافتن اقلام ۳ تایی پرتکرار بکار می‌روند و بصورت کلی، اقلام پرتکرار n تایی از ترکیبات اقلام پرتکرار n-1 تایی بدست می‌آیند.

البته تمام اقلام با هم ترکیب نمی‌شوند، بلکه با بکارگیری اصل Apriori از تولید ترکیبات بیهوده جلوگیری می‌کنیم: « یک مجموعه اقلام n تایی تنها زمانی می‌تواند پرتکرار باشد که تمام زیرمجموعه‌های n-1 تایی آن پرتکرار باشند». بر این اساس اگر دو مجموعه قلم مانند AB و AC پرتکرار باشند، ترکیب آنها یعنی ABC در صورتی می‌تواند پرتکرار باشد که مجموعه اقلام BC نیز پرتکرار باشد. زمانی که تمام n-1 زیرمجموعه پرتکرار باشند، ترکیب جدولهای در هم ساز دو تا از آنها برای تولید جدول در هم ساز مجموعه n تایی کافی است.

ساختار بکار رفته در الگوریتم پیشنهادی این امکان را ایجاد می‌کند که مجموعه داده‌ها فشرده و قابل نگهداری در حافظه اصلی گردند و بدین ترتیب نیاز به مراجعات پیاپی به دیسک و عملیات ورودی-خروجی پرهزینه مرتفع گردد.

البته لازم به ذکر است که این مزیت مهم در ارتباط با مجموعه داده‌هایی مشابه داده‌های سبد خرید که دارای یک طبیعت خلوت می‌باشند (احتمال مقدار ۱ بسیار کمتر از ۰ است)، صادق است و در مورد مجموعه داده‌هایی که احتمال مقدار ۱ در آنها با ۰ تفاوت چندانی ندارد، فشرده سازی صورت نمی‌گیرد و بنابراین در هر مرحله از الگوریتم مجبوریم تنها مقادیر مورد محاسبه را در حافظه نگه داریم.

۱-۳. تحلیل داده‌های سبد خرید

الگوریتم معرفی شده الگوریتمی کارا و مناسب برای مساله‌ای مانند تحلیل داده های سبد خرید و کشف مجموعه اجناسی است که به دفعات زیاد بطور همزمان (در یک سبد خرید) توسط مشتری ها

---

1. *Shift-Right*
2. *Shift-Left*
3. *Carry bits*

خریداری شده اند. در این مساله معمولاً یک پایگاه داده تحلیلی بزرگ از تراکنشها در اختیار داریم که هر تراکنش یک سبد خرید را نشان می‌دهد، یعنی شامل لیست اجناسی از فروشگاه است که توسط یک مشتری در یکبار مراجعه خریداری شده است. نمونه ای از این نوع داده‌ها در شکل ۱.(الف) مشاهده می‌شود. قبل از اعمال الگوریتم بر روی داده‌ها، بعنوان یک مرحله پیش پردازشی ابتدا مجموعه داده‌ها را به شکل ساخت‌یافته به طریقی که قبلا بیان شد (مانند شکل ۱.(ب)) تبدیل می‌کنیم.

در ساختار جدید، هر ستون نماینده یک قلم از تمام اجناس موجود در فروشگاه می‌باشد و هر ستون نشان‌دهنده وجود (۱) یا عدم وجود (۰) یک قلم جنس در یک سبد خرید است. همانطور که قبلاً هم بیان شد، همانطور که قبلا هم بیان شد، با توجه به اینکه تنوع اقلام یک فروشگاه معمولاً زیاد و برعکس تعداد اقلام موجود در یک سبد خرید معمولاً کم می‌باشد، نسبت تعداد ۱ها به صفرها در یک ستون عدد کوچکی است. بنابراین احتمال وجود قطعه‌های تماما صفر زیاد است که این عامل سبب کوچک بودن جدولهای در هم ساز و درنتیجه کارایی قابل قبول عملیات جستجو توسط الگوریتم پیشنهادی می‌گردد.

۳-۲. تعیین حد آستانه پشتیبانی

نکته مهمی که در مورد کاوش اقلام پرتکرار بایستی مورد توجه قرار گیرد، تعیین مقداری مناسب برای حداقل درجه پشتیبانی است. معمولاً زمانی که بحث از پرتکرار بودن یک قلم در یک مجموعه داده می‌شود، درجه پشتیبانی ۵۰ درصد به بالا در ذهن تداعی می‌شود. اما در مجموعه داده‌های سبد خرید (و سایر مجموعه داده‌های خلوت نظیر آن) با توجه به احتمال بسیار کم حضور یک قلم داده در یک تراکنش (بدلیل تنوع زیاد اقلام)، درجه پشتیبانی اقلام مختلف بسیار پایین‌تر از این مقدار می‌باشد. بعنوان مثال، فرض کنید فروشگاهی ۱۰۰۰۰ جنس مختلف دارد. احتمال خریداری شدن هریک از اجناس فروشگاه در یک سبد خرید بطور متوسط ۰/۰۱ ٪ است.

بنابراین اگر یک قلم جنس در ۲ ٪ کل خریدها توسط مشتریان خریداری شده باشد (پشتیبانی ۲ ٪ داشته باشد)، آن جنس می‌تواند پرتکرار تلقی شود و کشف آن می‌تواند مفید باشد. لازم بذکر است که با در نظرگرفتن اقلام بصورت ترکیبی، احتمال خریداری شدن همزمان چند قلم داده کمتر از ۰/۰۱ ٪ و با افزایش تعداد اقلام موجود در ترکیب، این احتمال بمراتب کمتر هم خواهد شد.

با توجه به آنچه که ذکر شد، حد آستانه مورد قبول برای درجه پشتیبانی در هنگام کار با مجموعه داده‌هایی نظیر داده‌های سبد خرید بایستی عدد کوچکی اختیار شود. معیارهای مناسبی که می‌توان برای تعیین این حد آستانه از آنها کمک گرفت، یکی تعداد کل اقلام داده مختلف (کل اجناس فروشگاه) و دیگر متوسط اندازه تراکنش‌ها (متوسط تعداد اجناس خریداری شده در سبدهای خرید) می‌باشد. نتایج آزمایش‌هایی که بر روی مجموعه داده‌های واقعی اعمال شده‌اند، بخوبی نشانگر پایین بودن حدود مقادیر پشتیبانی می‌باشند.

۳-۳. مقدار مناسب k

یک مساله مهم در هنگام بکارگیری الگوریتم، مشخص کردن اندازه هر قطعه (k) می‌باشد. اگر قطعه‌ها کوچک اختیار شوند، احتمال پیدا شدن قطعه‌های تماماً صفر که قرار است دور ریخته شوند، بیشتر می‌شود. اما از طرف دیگر تعداد عناصر جدولهای در هم ساز افزایش می‌یابد و در نتیجه برای ترکیب دو جدول در هم ساز به عملیات *AND* بیشتری نیاز است. اما هرچه k بزرگتر باشد، احتمال قطعه‌های تماماً صفر کمتر می‌شود ولی در مجموع عملیات *AND* کمتری مورد نیاز است.

ما از طریق آزمایش به این نتیجه رسیدیم که انتخاب مقادیر تا حد ممکن بزرگ برای k بهترین نتایج را در بر دارد که با توجه به دلایل زیر معقول هم بنظر می‌رسد. اگرچه مقادیر کوچک k منجر به پیدا شدن قطعه‌های زائد بیشتری می‌شود، اما اندازه قطعه‌هایی که دور ریخته می‌شوند بزرگ نیست و تاثیر آنچنانی بر کارایی ندارد. علاوه بر این، دور ریختن قطعه‌های تماماً صفر تنها راه کاهش دفعات شمارش نیست. در بسیاری از موارد، بیت‌های یک قطعه تماماً صفر نیستند، اما کد دودویی مرتبط با آن قطعه شامل یک گروه بیت صفر در یکی از دو انتهای آن است.

از آنجا که ما از عملیات *SHR* یا *SHL* به روشی که ذکر شد برای شمارش بیت‌های ۱ استفاده می‌کنیم، گروه بیت‌های صفری که در یکی از کناره‌ها واقعند در شمارش به حساب نمی‌آیند. برای مثال، فرض کنید یکی از مقادیری که در یک جدول در هم ساز وارد شده عدد ۱۶۰ باشد.

با فرض اینکه اندازه قطعه‌ها برابر با ۸ اختیار شده باشد، کد دودویی معادل این عدد برابر ۱۰۱۰۰۰۰۰ می‌باشد. برای شمردن تعداد بیت‌های ۱ این عدد تنها ۳ عمل *SHL* انجام می‌شود. به عبارت دیگر، ۵ بیت صفری که در کناره سمت راست قرار دارند، اصلا خوانده نمی‌شوند. نتیجتاً مقدار k بهتر است تا حد ممکن بزرگ انتخاب شود. از آنجا که ما می‌خواهیم بیت‌های موجود در هر قطعه را در قالب یک عدد صحیح (دهدهی) نگه داریم (تا عملیات AND طی یک مرحله قابل انجام باشد)، حداکثر طول هر قطعه برابر با تعداد بیت مورد نیاز برای نمایش بزرگترین عدد صحیح در دامنه مقادیر صحیح می‌باشد. بنابراین، بالاترین حد ممکن برای k به معماری کامپیوتر و محیط برنامه نویسی بستگی دارد، که با توجه به دامنه اعداد صحیح می‌تواند مثلا ۳۲ و یا ۶۴ باشد. ما در تمام آزمایش‌ها، مقدار k را برابر ۳۲ قرار دادیم تا مجموعه بیت‌های هر قطعه (۳۲ بیت) بتوانند معادل یک عدد صحیح باشند.

۴. نتایج عملی مقایسه الگوریتم‌ها

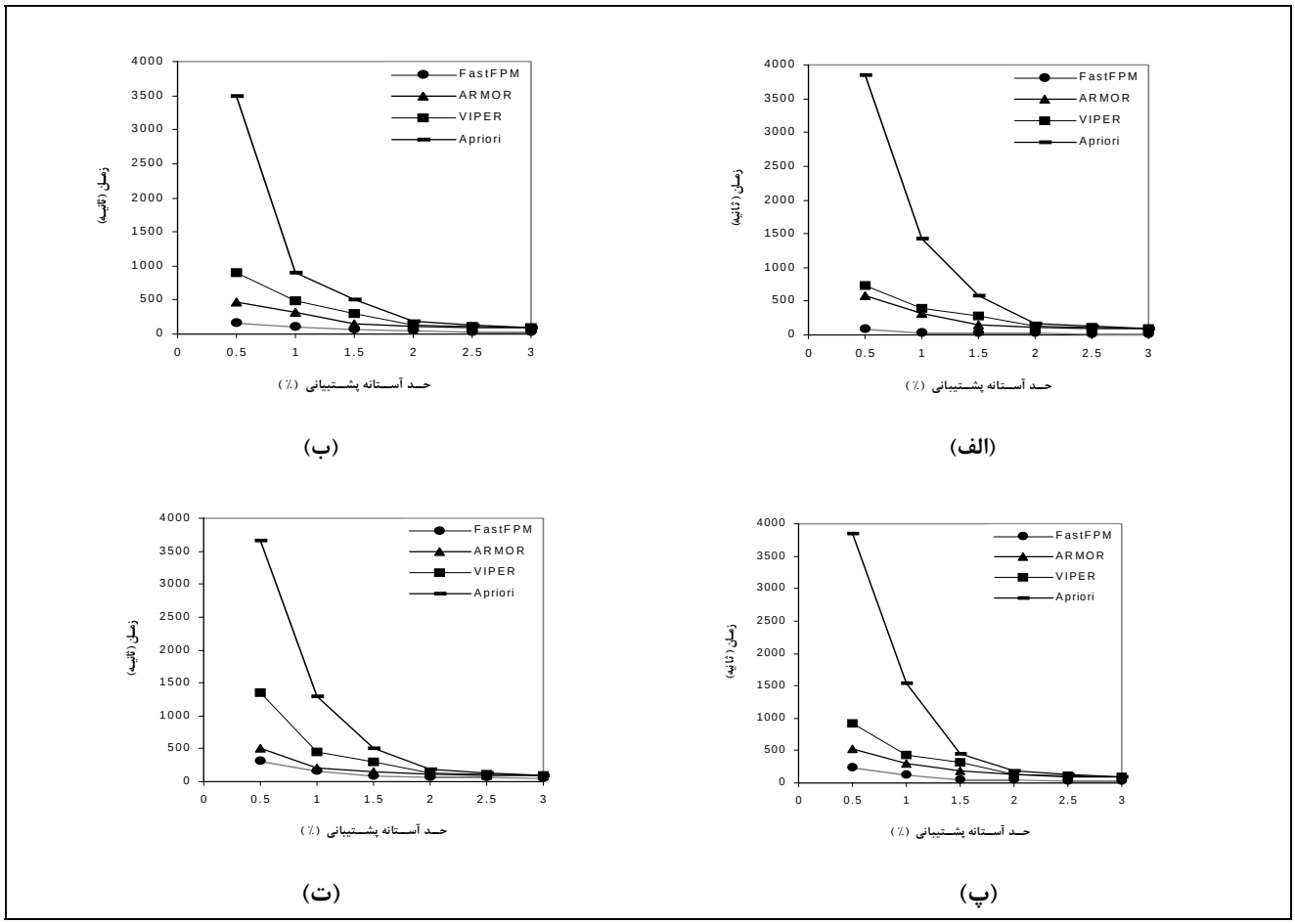
برای ارزیابی الگوریتم پیشنهادی و مقایسه آن با الگوریتمهای موجود دو مدل آزمایش ترتیب دادیم. در حالت اول، آزمایش بر روی داده‌های ساختگی و در حالت دوم روی داده‌های واقعی انجام شد. همانطور که می‌دانیم الگوریتمهای کاوش قوانین تداعی عموماً روال مشخصی را طی می‌کنند و در شرایط مشابه خروجی‌های یکسانی دارند.

بنابراین، معیار مهم در مقایسه روش‌های مختلف کارایی آنها از لحاظ زمان پاسخ و حافظه مورد نیاز می‌باشد[۲۴]. لذا، در این آزمایش‌ها، کارایی الگوریتم پیشنهادی (با در نظر گرفتن $k = 32$) را با الگوریتم‌های *Apriori* ، *VIPER* و *ARMOR* که از معروف‌ترین و کاراترین الگوریتم‌های موجود هستند، مقایسه کردیم. الگوریتم معروف دیگری به نام *FP-growth* نیز وجود دارد که بدلیل اینکه برای مجموعه داده‌های خیلی حجیم اجرای آن نیاز به حافظه بسیار زیادی دارد، تنها توانستیم آن را برای مجموعه داده‌های واقعی که کوچک‌تر هستند، ارزیابی کنیم. پیاده سازی الگوریتم‌ها را با زبان C++ انجام و بر روی یک سیستم با حافظه اصلی ۱ گیگا بایت و سرعت پردازنده ۳ گیگا هرتز اجرا شده‌اند.

۴-۱. آزمایش ۱: داده‌های تولید شده تصادفی

در آزمایش اول، الگوریتمهای نامبرده (بجز *FP_Growth*) را به همراه الگوریتم پیشنهادی بر روی ۱۰ مجموعه داده تصادفی هریک شامل ۲ میلیون تراکنش و ۵۰۰ قلم داده، بازاء هشت حد آستانه مختلف برای پشتیبانی اجرا کردیم. این آزمایش ۴ مرتبه تکرار شد و مجموعه داده‌ها در مراحل مختلف طوری تصادفی تولید شدند که احتمال بیت ۱ (وجود یک قلم داده در یک تراکنش) به ترتیب ۰/۰۰۵ ، ۰/۰۱ ، ۰/۰۵ و ۰/۱ باشد. نتایج این آزمایش در نمودار شکل۵ نشان داده شده‌است. محور x در این نمودار نشان دهنده مقادیر مختلف حد آستانه پشتیبانی است و محور y نمایانگر زمان اجرا بر حسب ثانیه است. مقادیری که برای زمان اجرای هر الگوریتم در نمودار مشاهده می‌شود، میانگین زمان اجراها برای ۱۰ مجموعه داده مختلف است. جدول۱ میزان حافظه مصرفی الگوریتم (جهت نگهداری جدولهای در هم ساز) را در هر مرحله از آزمایش نشان می‌دهد. از مقایسه این مقادیر با اندازه مجموعه داده اولیه (تقریبا ۱۰۰ مگابایت) می‌توان به میزان فشرده سازی الگوریتم پی برد.

با توجه به شکل۵ می‌بینیم که یک برتری قابل توجه برای کارایی الگوریتم *FastFPM* در مقایسه با *Apriori* و *VIPER* و یک برتری نسبی در مقایسه با *ARMOR* وجود دارد.



شکل۵. کارایی روش‌های مختلف بر روی داده‌های تصادفی با احتمال‌های مختلف برای بیت ۱ در داده‌ها

(الف) ۰/۰۰۵ (ب) ۰/۰۱ (پ) ۰/۰۵ (ت) ۰/۱

با مقایسه بخش‌های مختلف شکل۵ می‌توان دریافت که هرچه مجموعه داده خلوت‌تر باشد (نسبت تعداد یک‌ها به صفرها کمتر باشد)، برتری الگوریتم پیشنهادی بر سایر روش‌ها نمایان‌تر می‌شود. این پدیده را می‌توان اینطور توجیه کرد که بازاء مجموعه داده‌های خلوت احتمال پیدایش قطعه‌های تماماً صفر بیشتر بوده و در نتیجه اندازه جدولهای در هم ساز به مراتب کوچکتر خواهد بود. بدین ترتیب سرعت عملیات محاسبه پشتیبانی اقلام نیز کاهش می‌یابد. نتایج نشان داده شده در جدول۱ را می‌توان گواهی بر این ادعا دانست.

همچنین در شکل۵ مشاهده می‌شود که زمان اجرای الگوریتم‌ها با کم شدن حد آستانه پشتیبانی بصورت نمایی افزایش می‌یابد. این مساله با توجه به اینکه تعداد مجموعه اقلام پرتکرار کشف شده با کم شدن حد آستانه پشتیبانی بطور نمایی افزایش می‌یابد، قابل توجیه است.

جدول ۱. مقدار حافظه مصرفی در مراحل مختلف آزمایش ۱

احتمال بیت ۱ در داده‌ها	حافظه مصرفی (مگابایت)
۰/۰۰۵	۶/۸
۰/۰۱	۱۱/۳
۰/۰۵	۴۷/۶
۰/۱	۸۸/۷

۲-۴. آزمایش ۲: مجموعه داده‌های واقعی

آزمایش دوم ما بر روی چند مجموعه داده واقعی به نامهای BMS-POS، BMS-WebView-1 و BMS-WebView-2 انجام شد. BMS-POS مجموعه‌ای از داده‌های مربوط به اقلام فروخته شده یک فروشگاه لوازم الکترونیکی بزرگ است که طی سال‌ها جمع آوری شده‌است. اقلام داده در این مجموعه همان اجناس فروشگاه هستند و اقلام خریداری شده مشتری‌ها در هر مراجعه یک تراکنش را تشکیل می‌دهند. هدف از جمع آوری این داده‌ها یافتن ارتباطاتی بین اقلام خریداری شده بوده‌است. این مجموعه داده مشتمل بر ۵۱۵,۵۹۷ تراکنش و ۱۶۵۷ قلم داده مختلف است. مجموعه‌های BMS-WebView-1 و BMS-WebView-2 هم شامل داده‌هایی مربوط به مراجعات کاربران طی چند ماه به سایت‌های دو فروشگاه اینترنتی هستند که بر اساس محصولاتی که کاربر در هر بار مراجعه به سایت جهت بازدید بر روی آنها کلیک کرده‌است، تهیه شده‌اند و در واقع هر تراکنش شامل محصولاتی است که کاربر در یک بار مراجعه به سایت بازدید کرده‌است. هدف از جمع آوری این داده‌ها هم یافتن ارتباطات بین اقلام بر اساس بازدیدهای همزمان است. این دو مجموعه داده بترتیب شامل ۵۹,۶۰۲ و ۷۷,۵۱۲ تراکنش هستند و تعداد اقلام داده مختلف آنها به‌ترتیب ۴۹۷ و ۳۳۴۰ مورد می‌باشد. در این آزمایش‌ها حد آستانه MinConf را صفر در نظر گرفته و معیار MinSupp را در ارزیابی کارایی الگوریتم‌ها اعمال کردیم.

زمان اجرای الگوریتم‌های مختلف برای کشف مجموعه اقلام پرتکرار و تولید قانون بر روی سه مجموعه داده، بازاء مقادیر مختلف MinSupp اندازه گیری و در شکل‌های ۶.(الف) تا ۶.(پ) نمایش داده شده‌است. همانطور که در شکل‌ها دیده می‌شود، زمان اجرای الگوریتم پیشنهادی (*FastFPM*) بازاء مقادیر کوچکتر MinSupp حداقل نصف روش *ARMOR* که از بقیه روش‌های مورد بحث کاراتر است، می‌باشد.

۵. نتیجه گیری

به دلیل اهمیت و کاربرد زیاد قوانین تداعی، روش‌های زیادی جهت کشف این قوانین از بین داده‌های انبوه پیشنهاد شده‌اند. بخش عمده اکثر الگوریتم‌های تولید قانون، فرایند کشف مجموعه اقلام پرتکرار است. هر روشی که پیشنهاد شده سعی در بهتر کردن روش‌های قبلی دارد. رویکرد اکثر روش‌ها بر این بوده که تا حد ممکن نیاز به پیمایش‌های مکرر کل داده‌های دیسک جهت یافتن اقلام ترکیبی پرتکرار کاسته شود و در صورت امکان کشف این اقلام بطور مستقیم صورت گیرد.

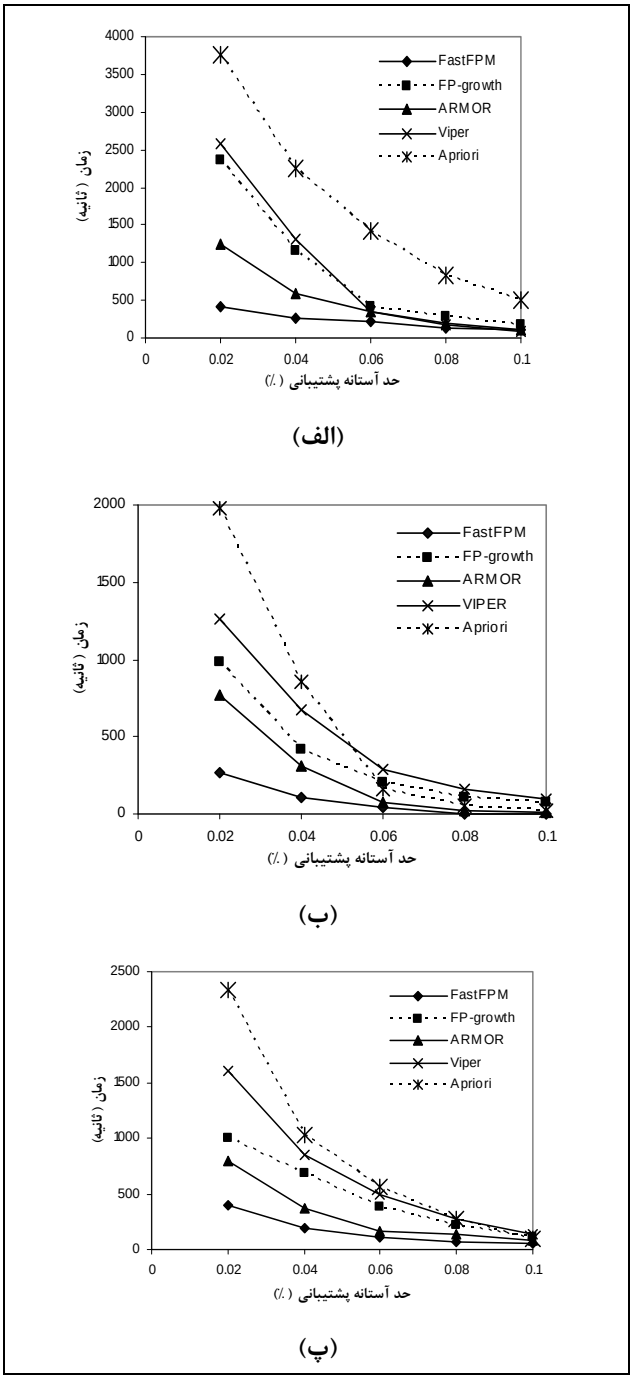
با وجود اینکه روش‌های کارآمدی در این راستا پیشنهاد شده‌اند، اما در اغلب موارد راهی بجز شمردن دفعات تکرار اقلام جهت محاسبه پشتیبانی آنها وجود ندارد. در این مقاله تمرکز ما بر این بوده که اولا بطریقی بهینه کل داده‌ها را در حافظه تگه داشته و از مراجعات پرهزینه به دیسک خودداری کنیم و ثانیاً راه حلی برای شمارش دفعات تکرار اقلام ترکیبی بصورت کارا ارائه دهیم. در روش پیشنهادی خواندن داده‌ها از دیسک تنها یک بار انجام می‌شود و بعد از آن داده‌ها بصورت کد شده در یک ساختار بخصوص در حافظه نگهداری می‌شوند. برای ارزیابی کارایی این روش و مقایسه آن با روش‌های دیگر دو مدل آزمایش بر روی داده‌های ساختگی و داده‌های واقعی ترتیب داده و زمان‌های اجرای هر الگوریتم را در شرایط مختلف مشاهده کردیم.

دلایل عمده برتری الگوریتم ما از لحاظ کارایی را می‌توان در عدم نیاز به خواندن پیاپی داده‌ها از دیسک، کاهش فضای جستجو در هر مرحله (با دور ریختن بخش‌های زائد) و استفاده از عملیات سریع بیتی در حافظه دانست. البته شرط اینکه روش پیشنهادی از کارایی بالایی برخوردار باشد این است که مجموعه داده‌ها همانند داده‌های مساله سبد خرید ذاتاً خلوت باشد، یعنی احتمال وجود یک قلم داده در کل مجموعه کم باشد. در چنین شرایطی بخش‌های زیادی از داده‌ها (شامل توالی‌های صفر) دور ریخته شده و قسمتهای با ارزش بصورت کد شده در حافظه نگهداری می‌شوند.

شرط ذکر شده در مجموعه داده‌هایی از قبیل سبدهای خرید بدلیل تنوع اقلام و نیز محدود بودن متوسط تعداد اقلام سبدها

معمولاً برقرار است. در این شرایط، با توجه به احتمال بسیار کم حضور یک قلم داده در یک تراکنش، محدوده درجه پشتیبانی اقلام مختلف بسیار پایین می باشد.

بنابراین، همانطور که در آزمایش‌ها نیز ملاحظه شد، حد آستانه مورد قبول برای درجه پشتیبانی در هنگام کار با این نوع داده‌ها معمولاً عدد کوچکی انتخاب می‌شود.



شکل۶. زمان اجرای الگوریتم‌های مختلف بر روی مجموعه داده‌های واقعی: (الف) BMS-POS (ب) BMS-WebView-1 (پ) BMS-WebView-2

مراجع

[1] Brin, S., Motwani, R., Silverstein, C., *Beyond Market Baskets: Generalizing Association Rules to Correlations*, In Proc. of the 1997 ACM SIGMOD international conference on Management of data, Tucson, Arizona, United States, 1997, pp.265-276.

[2] Taheri, M., Boostani, R., *Novel Auxiliary Techniques in Clustering*, In Proc. of World Congress on Engineering (WCE2007), pp. 173-178, London, UK, 2-4 July 2007.

[3] Fakhrahmad, S.M., Zolghadri Jahromi, M., *Constructing Accurate Fuzzy Classification Systems: A New Approach Using Weighted Fuzzy Rules*, In Proc. of 4th International Conference (IEEE) on Computer Graphics, Imaging and Visualization, (CGIV07), Bangkok, Thailand, 15-17 August 2007, pp. 408-413.

[4] Fakhrahmad, S.M., Sadreddini, M.H., Zolghadri Jahromi, M., *AD-Miner: A New Incremental Method for Discovery of Minimal Approximate Dependencies Using Logical Operations*, Intelligent Data Analysis, Vol. 12 , No. 6 (In press), 2008.

[5] Lin, J., Dunham, M. H., *Mining Association Rules: Antiskew Algorithms*, In Proc. of Intl. Conf. on Data Engineering (ICDE), 1998, pp. 128-134.

[6] Pudi, V., Haritsa, J., *Quantifying the Utility of the Past in Mining Large Databases*, Information Systems, July 2000.

[7] Pudi V., Haritsa J., *On the Optimality of Association-Rule Mining Algorithms*, Technical Report TR-2001-01, DSL, Indian Institute of Science, 2001.

[8] Agrawal, R., Srikant, R., *Fast Algorithms for Mining Association Rules in Large Databases*, In Proc. of the 20th International Conference on Very Large Data Bases, 1994, pp. 487-499.

[9] Shenoy, P., Haritsa J., Sudarshan S., Bhalotia G., Bawa M., and Shah D., *Turbo-Charging Vertical Mining of Large Databases*. In Proc. of ACM SIGMOD Intl. Conf. on Management of Data, ACM Press, Vol. 29 , No. 2, 2000, pp. 22 - 33.

[10] Pudi, V., Haritsa, J.R., *ARMOR: Association Rule Mining Based on Oracle*. In Proc. of ICDM Workshop on Frequent Itemset Mining Implementations, pp. 131-139, Florida, USA, 2003.

[11] Han, J., Pei, J., and Yin Y., *Mining Frequent Patterns Without Candidate Generation*. In Proc. of ACM SIGMOD Intl. Conf.on Management of Data, ACM Press, Vol. 29, No. 2, 2000, pp. 1 – 12.

[12] Pei, J., Han, J., Mao R., *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, In Proc. of ACM_SIGMOD International workshop on Data Mining and Knowledge Discovery, 2002, pp. 365-372.

[13] Dong, G., Li, J., *Efficient mining of Emerging Patterns: Discovering Trends and Differences*, In Proc. of the fifth ACM SIGKDD international conference on Knowledge

discovery and data mining, San Diego, California, United States, 1999. pp.43-52.

[14] Silverstein, C., Brin, S., Motwani, R., Ullman, J.D., *Scalable Techniques for Mining Causal Structures*, In Proc. of the 24rd International Conference on Very Large Data Bases, 1998, pp.594-605.

[15] Mannila, H., Toivonen, H., Verkamo, A. I., *Discovery of Frequent Episodes in Event Sequences*, Data Mining and Knowledge Discovery, Vol. 1, No.3, 1997, pp.259-289.

[16] Savasere, A., Omiecinski, E., Navathe, S., *An Efficient Algorithm for Mining Association Rules in Large Databases*, In Proc. of Intl. Conf. on Very Large Databases (VLDB), 1995. pp. 117-125.

[17] Zaki, M. J., Gouda, K., *Fast vertical Mining Using Diffsets*, Technical Report 01-1, Rensselaer Polytechnic Institute, 2001.

[18] Webb, G.I., *OPUS: An efficient Admissible Algorithm for Unordered Search*, JAIR, Vol. 3, 2000, pp. 431-465.

[19] Liu, B., Hsu, W. and Ma, Y., *Pruning and Summarizing the Discovered Associations*. In Proc. of the Fifth ACM_SIGKD International Conference on Knowledge Discovery and Data Mining, New York, NY: ACM, 125-134, 1999.

[20] Webb, G.I., *Efficient Search for Association Rules*. In Proc. of the Sixth ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY: ACM, 2000, pp. 99-107.

[21] Zaki, M.J., *Generating Non-Redundant Association Rules*, In Proc. of the Sixth ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY: ACM, 2000, pp. 34-43.

[22] Srikant, R., Agrawal, R., *Mining Generalized Association Rules*, In Proc. of Intl. Conf. on Very Large Databases (VLDB), Sept. 1995.

[23] Zheng, Z., Kohavi, R. and Mason L., *Real World Performance of Association Rule Algorithms*. In Proc. of Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 2001, pp. 421-428.

[24] Xiao, Y., Dunham, M. H., *Considering Main Memory in Mining Association Rules*, In Proc. of Intl. Conf. on Data Warehousing and Knowledge Discovery (DAWAK), 1999, pp. 289-296.