https://doi.org/10.33261/jaaru.2021.28.1.008 *Association of Arab Universities Journal of Engineering Sciences (2021)28(1):83–98*

Association of Arab Universities Journal of Engineering Sciences
مجلة اتحاد الجامعات العربية للدراسات والبحوث الهندسية

التنقيب عن العناصر المتكررة بالاعتماد على تطوير خوارزمية **FP-growth** واستخدام تقنية **MapReduce**

زكريا مهروسة[1]، ديما مفتي الشوافعة[2]، حسن قزاز[3]*

[1] قسم هندسة الحواسيب، كلية الهندسة الكهربائية والالكترونية، جامعة حلب، سوريا، Rmahrousa@hotmail.com.

[2] قسم هندسة الحواسيب، كلية الهندسة الكهربائية والالكترونية، جامعة حلب، سوريا، dima.mufti@gmail.com.

[3] قسم هندسة الحواسيب، كلية الهندسة الكهربائية والالكترونية، جامعة حلب، سوريا، kazzazhasan@gmail.com.

* الباحث الممثل: حسن قزاز، kazzazhasan@gmail.com.

نشر في: *31 اذار 2021*

الخلاصة – إيجاد قواعد الترابط (Association Rules) في البيانات الضخمة (Big Data) عملية مهمة في مجال التنقيب عن البيانات (Data Mining) واكتشاف المعرفة (Knowledge Discovery). تنتج البيانات الضخمة من النمو اليومي المضطرد (الأسي) للبيانات، يمكن أن نواجه العديد من التحديات عندما نقوم بعملية التنقيب عن العناصر المتكررة (Frequent Itemset Mining FIM) في البيانات الضخمة مثل المساحة الذاكرية المطلوبة، تعدد الأبعاد للبيانات، تجانس البيانات، إلخ. يمكن تخفيض درجة تعقيد عملية التنقيب عن العناصر المتكررة باستخدام خوارزمية FP-growth مُطورة وتوزيع عملية التنقيب من خلال البنية Map Reduce على أكثر من حاسب في Hadoop. تم في هذا البحث اقتراح خوارزمية مُطورة عن خوارزمية FP-growth بالاعتماد على المخطط الموجه (Directed Graph) والهادوب من أجل تخفيض الزمن اللازم لإيجاد العناصر المتكررة ضمن قواعد البيانات الضخمة، والعمل على توزيع هذه المهمة على أكثر من حاسب. تم اختبار الخوارزمية المقترحة على قواعد بيانات قياسية، وأثبتت النتائج فعالية الخوارزمية وقدرتها على التعامل مع قواعد البيانات الضخمة والقيام بعملية التنقيب فيها. بالإضافة إلى الحصول على تحسن كبير في معدل استهلاك الذاكرة من حيث تخزين العناصر المتكررة، بالإضافة إلى التعقيد بالنسبة إلى الزمن.

الكلمات الرئيسية – التنقيب عن قواعد الترابط، البيانات الضخمة، Map Reduce، FP-growth، المخطط الموجه.

1. المقدمة:

التنقيب عن البيانات هو عملية اكتشاف المعلومات في مجموعة ضخمة من البيانات، والهدف الأساسي من هذه العملية هو تحليل تلك البيانات من أجل استخراج نماذج وقواعد مهمة منها لم تكن معروفة بشكل مسبق، ولا يمكن استكشاف تلك النماذج بالطرق التقليدية لأسباب عديدة: منها أن العلاقات بين البيانات قد تكون معقدة جداً أو بسبب ضخامة تلك البيانات. يمكن تقسيم تقنيات التنقيب عن البيانات إلى عدة محاور بحسب الطريقة المتبعة في اكتشاف المعرفة. حيث يوجد تقنيات التنقيب عن قواعد الترابط (Association Rules Mining ARM)، التصنيف (Classification)، التجميع (Clustering)، إلخ[11] . تم اقتراح العديد من الخوارزميات الفعالة والقابلة للتطوير لاستخدامها في إيجاد الترابط بين النماذج المتكررة وقواعد الترابط، أشهر تلك الخوارزميات هي خوارزمية (Apriori Algorithm) [1] [19]و(Eclat Algorithm) [16] و (Frequent Pattern FP-growth) [7] [8]. كل خوارزمية من الخوارزميات السابقة تمتلك بعض السلبيات والتي تلخص بشكل عام كمايلي: توليد عدد كبير من مجموعات العناصر المرشحة، وضرورة المسح المتكرر لقاعدة البيانات، والاستهلاك الذاكري الكبير، بالإضافة إلى بطيء تلك الخوارزميات وخصوصاً عند التعامل مع أحجام بيانات ضخمة مما يؤدي إلى فشل عملية التنقيب.

لذلك تم العمل على إيجاد استراتيجيات لمعالجة المشاكل السابقة ومنها الحوسبة السحابية (Cloud Computing)، فمع ظهور هذا المفهوم تم حل العديد من المشاكل التي تواجه الخوارزميات التقليدية للتنقيب عن العناصر المتكررة وإيجاد قواعد الترابط، خصوصاً عند التعامل مع قواعد البيانات الضخمة [4][6].

حيث يمكن التغلب على مشاكل التنقيب التسلسلي عن العناصر المتكررة من خلال الدمج بين مفهوم الحوسبة السحابية والتنقيب عن العناصر المتكررة.

2. هدف البحث:

إن إيجاد العناصر المتكررة مرحلة مهمة في عملية التنقيب عن قواعد الترابط والهدف الأساسي لهذا البحث هو تطوير خوارزمية FP-growth لتحسين أدائها وإيجاد حلّ للسلبيات الناتجة من عملية التنقيب عن العناصر المتكررة من ناحية الزمن والمساحة التخزينية الكبيرة اللازمة لعملية التنقيب وخصوصاً عند التعامل مع البيانات الضخمة. بالإضافة الى تجاوز العديد من سلبيات الخوارزميات السابقة. حيث سيتم الاعتماد على استخدام إطار تخزين ومعالجة البيانات الموزعة هادوب لبناء خوارزمية تفرعية (**P**arallel **G**raph **F**requent **P**attern PGFP-growth)، معتمدة على المخطط الموجه لضغط قاعدة البيانات الضخمة بشكل كبير من أجل توفير الزمن اللازم لعملية التنقيب والمساحة التخزينية المطلوبة، وتخفيض التعقيد الحسابي والاتصال بين عقد التجمع بالإضافة إلى تحليل ومقارنة النتائج مع الخوارزميات السابقة.

3. مفهوم إيجاد العناصر المتكررة:

إيجاد العناصر المتكررة مرحلة أساسية في خوارزميات التنقيب عن العناصر المتكررة، يمكن أن تُوصف عملية إيجاد العناصر المتكررة كما يلي: إذا كان لدينا مجموعة من العناصر $I=\{I_1,I_2,I_3,\ldots\ldots I_n\}$ حيث n: تمثل العدد الكلي للعناصر، واذا كان لدينا مجموعة من المداولات (Transactions) ضمن قاعدة البيانات $DB=\{T_1,T_2,\ldots,T_m\}$ حيث m: تمثل عدد المداولات الكلي علماً أن T_i $(i \in [1..m])$ ، كل مداولة تحوي مجموعة من العناصر المنتمية إلى I. نسمي المجموعة الجزئية $X \subseteq I$ بمجموعة عناصر (Itemset) أو نموذج (Pattern). كما يسمى النموذج X يسمى بـ k-itemset أي يحوي K عنصر يتكررا معاً في المداولات. وتعرف قيمة الدعم للنموذج X والتي يرمز لها بـ sup(X) بأنها عدد المداولات التي تتضمن العنصر X. ونقول عن العنصر X أنه عنصر مكرر إذا وفقط إذا كان الدعم له sup(X) أكبر أو يساوي الدعم الأصغري (قيمة عددية أكبر من الصفر معرفة من قبل المستخدم ويرمز لها بـ min_sup). على سبيل المثال إذا كان X يحوي عنصرين I_1، I_2 قيمة الدعم له 4 فهذا يعني أن العنصرين قد تكرر وجودهما معاً في 4 مداولات. أما قواعد الترابط فتأخذ الشكل $X \Longrightarrow Y$، حيث إن

$X \subseteq I$ ، $Y \subseteq I$، $X \neq \emptyset$، $Y \neq \emptyset$، $X \cap Y \neq \emptyset$، ومقدار الثقة (Confidence) لأي قاعدة هو $\frac{\sup(X \cup Y)}{\sup(X)}$. والمشكلة الأساسية في عملية التنقيب عن قواعد الترابط، هو إيجاد جميع القواعد التي تمتلك قيمة دعم أكبر من قيمة الدعم الأصغري min_sup ومقدار ثقة أكبر من عتبة الثقة الأصغرية أيضاً[7] [8] [16] .

4. الدراسة المرجعية والأبحاث السابقة:

هناك عدة خوارزميات تختص بإيجاد الترابط بين البيانات من خلال إيجاد النماذج المتكررة وقواعد الترابط، في هذه الفقرة سوف نناقش الخوارزميات الأساسية للتنقيب عن العناصر المتكررة، ثم الخوارزميات الخاصة بالبيانات الضخمة.

4.1 *الخوارزميات الأساسية للتنقيب عن البيانات:*

يمكن تقسيم الخوارزميات الأساسية للتنقيب عن العناصر المتكررة حسب عدد مرات عبور أو مسح قاعدة البيانات إلى:

- **الخوارزميات المعتمدة على العبور المتكرر لقاعدة البيانات:**

خوارزمية Apriori والتي تستخدم طريقة التوليد والاختبار Generate and Test، والعبور المتكرر لقاعدة البيانات، أي أنها تولد العناصر المرشحة ثم تختبر فيما إذا كانت تمثل تكراراً. هذه الخوارزمية مكلفة من حيث الزمن والمساحة التخزينية ناتجة عن استراتيجية البحث بالعرض Breadth) (First Search، خاصة إذا كانت قاعدة البيانات ضخمة وكان عدد العناصر المرشحة المختبرة كبيراً[1] .

أما خوارزمية Eclat تقوم على مبدأ Vertical Data Format تحويل الأسطر في الجدول إلى أعمدة، والعبور المتكرر لقاعدة البيانات، أي أنه لكل عنصر في قاعدة البيانات يتم تخزين قائمة بأرقام السجلات التي ينتمي اليها. ثم يتم حساب مجموعات العناصر المتكررة. تعتبر خوارزمية سريعة بسبب استراتيجية البحث في العمق (Depth First Search)، لكن القوائم الوسيطة التي تضم أرقام سجلات العناصر تكون ضخمة جداً في قواعد البيانات الضخمة[16] .

- **الخوارزميات المعتمدة على القيام بعبورين لقاعدة البيانات:**

خوارزمية FP-growth والتي تعتمد على فرضية فرّق تسد (Divide and Conquer) وتتمتع بالمزايا التالية: مسح قاعدة البيانات لمرتين فقط، تحويل قاعدة البيانات إلى بنية شجرية FP tree حجمها دائماً أصغر من حجم قاعدة البيانات الأصلية، لاتولد عناصر مرشحة كما هو الحال في خوارزمية Apriori، وتعتبر أسرع من الخوارزميات السابقة[7] [8] [14].

- **الخوارزميات المعتمدة على القيام بعبور وحيد لقاعدة البيانات:**

خوارزمية العبور الوحيد لشجرة CP-Tree، قدمت مثال عن بنية شجرية أمثليه (Compact Pattern Tree CP-Tree)، تقوم بمسح قاعدة المعطيات من خلال طور الإدخال (Insertion phase) وتطبق نفس الآلية المتبعة في خوارزمية FP-growth للقيام بعملية التنقيب (Restructuring phase)، بالرغم من حاجة هذه الخوارزمية لعملية مسح واحدة إلا أن زمن عملية إنشاء Header Table وبناء الشجرة يزداد مع زيادة حجم قاعدة المعطيات بشكل أسي[21] .

كل الخوارزميات التسلسلية السابقة تعاني من فشل عملية التنقيب عند التعامل مع قواعد البيانات الضخمة ذات الكثافة العالية. بسبب الزيادة الكبيرة في المساحة الذاكرية اللازمة لعملية المعالجة.

4.2 *الخوارزميات الخاصة بالبيانات الضخمة:*

يوجد العديد من الخوارزميات المطورة من أجل التعامل مع البيانات الضخمة ومعالجة العديد من المشكلات مثل: كلفة التواصل بين العقد الحسابية (Communication cost)، معدل عمليات الدخل والخرج (I/O utilization)، المساحة الذاكرية والزمن (space and memory)، قابلية التوسع (Scalability). تم تطبيق العديد من التقنيات الحديثة بهدف حلّ المشاكل السابقة مثل: Mapreduce، (**M**essage **P**assing **I**nterface) MPI، (**F**ield-**P**rogrammable **G**ate **A**rray) FPGA، وحدة معالجة الرسومياتGPU (**G**raphics **P**rocessing **U**nit) ، استخدام العناقيد الحسابية Cluster [3].

- **الخوارزميات المعتمدة على شريحة FPGAs:**

تم العمل على تطوير خوارزمية FP-growth تفرعية من خلال توزيع مهمة التنقيب عن العناصر المتكررة عبر شريحة مصفوفة البوابات القابلة للبرمجة FPGAذات بنية متعددة المعالجات (Multi-Processor System). تعمل الخوارزمية على تقسيم الشريحة إلى عدة وحدات معالجة Processing) ((PEs Entities يتم تقسيم الأحمال فيما بينها بشكل متساوي، إلا أنها تعاني من كمية البيانات الهائلة المتبادلة بين وحدات المعالجة وخصوصاً عند التعامل مع قواعد البيانات الضخمة حيث تستهلك الخوارزمية جزء كبير من دورة الساعة من أجل تبادل البيانات بين وحدات المعالجة [22].

- **الخوارزميات المعتمدة على وحدة معالجة الرسومياتGPUs :**

عملت العديد من الأبحاث على استخدام وحدة معالجة الرسوميات نظراً لفعاليتها وقدراتها الحسابية العالية في معالجة البيانات الموزعة.

خوارزمية Jiang H and Meng H.) Parallel FP-growth)، تعتمد على تطبيق خوارزمية FP-growth على وحدة معالجة الرسوميات GPU، والتي تتألف من آلاف الأنوية الحسابية (Computing Cores) القابلة للبرمجة. ونظراً لصعوبة تطبيق بنى المعطيات غير المنتظمة كالأشجار على وحدة معالجة الرسوميات تم إيجاد آلية لتحويل شجرة FP-Tree إلى مصفوفة ثنائية الأبعاد FP-array بشكل منطقي [10].

كما يوجد في البحث[11] ، خوارزمية CU-Aprior والتي تعتمد على طريقتين مختلفتين من أجل تفريع عملية توليد العناصر المرشحة، وحساب تكرار كل عنصر من خلال GPU.

النسخ الموزعة من خوارزمية Aprior وEclat المعتمدة على وحدة معالجة الرسوميات تتميز بفعاليتها وقدرتها على التعامل مع قواعد البيانات الضخمة وعند عتبات الدعم المنخفضة. على عكس النسخ الموزعة من خوارزمية FP-growth المعتمدة على وحدة معالجة الرسوميات، نتيجة صعوبة التعامل مع بنية الشجرة، يتم التغلب على هذه المشكلة من خلال تحويل البنية الشجرية إلى بنية أخرى مكافئة أو من خلال العمل على تفريع خوارزمية FP-growth حيث يتم الدمج بين GPUs وCPUs متعددة النوى (Multi-core).

- **الخوارزميات المعتمدة على العناقيد الحسابية CPUs:**

من أوائل الخوارزميات في هذا المجال، الخوارزمية المقترحة في المرجع [14] والتي اعتمدت على تفريع خوارزمية FP-growth الأساسية من خلال البنية MapReduce.

الخوارزمية المقدمة (Alhamodi A. et al.) MPFP-growth هي عبارة عن خوارزمية FP-growth تفرعية باستخدام البنية Map Reduce ضمن الأطار هادوب. لم يتم اختبار هذه الخوارزمية عند عتبات الدعم المنخفضة، كما أنها لم تحقق زيادة كبيرة في السرعة عند العتبات التي تزيد عن 50% [4].

خوارزمية (Spark FPGrowth) S-FPG للتنقيب عن النماذج المتكررة (Gassama et al.)، والتي تعتبر نسخة تفرعية عن خوارزمية FP-growth باستخدام إطار تخزين ومعالجة البيانات الموزعة Apache Spark™، تم اختبار الخوارزمية على عنقود مكون من أربع عقد حسابية، كل منها تحوي 8 نوى Intel® Xeon® E5 حيث حققت هذه الخوارزمية زيادة في السرعة بمقدار 6 أضعاف عن خوارزمية FP-growth تفرعية باستخدام البنية Map Reduce [6].

قام (Ji S. et al.) بتطوير خوارزمية FP-growth من خلال ما يسمى بالمسارات المشتركة (Sharing Path) وذلك من أجل التقليل من مسح قاعدة البيانات. استخدمت البنية Map Reduce من أجل الحساب التفرعي للعناصر المتكررة، وتمكين الخوارزمية المقترحة على التعامل مع قواعد البيانات الضخمة، تعطي الخوارزمية المطورة سرعة أكبر من خوارزمية FP-growth الأساسية وخوارزمية FP-growth تفرعية باستخدام البنية Map Reduce عند التعامل مع قواعد البيانات الضخمة إلا أنها لم تحقق زيادة ملحوظة في السرعة مع قواعد البيانات المتوسطة والصغيرة [9].

قدم (Kulkarni P G et al.) خوارزمية تفرعية معدلة عن خوارزمية Aprior باستخدام جداول الهاش (Hash Table)، حيث تستخدم الخوارزمية المقترحة البنية Map Reduce على ثلاث مراحل. في المرحلة الأولى تستخدم البنية Map Reduce من أجل إيجاد العناصر المتكررة لأول مجموعة(1-itemset) . أما المرحلة الثانية فتستخدم البنية Map Reduce من أجل حذف العناصر الغير متكررة والتي لا تحقق عتبة الدعم الدنيا. وفي المرحلة الأخيرة من البنية Map Reduce يتم استخدام خوارزمية Aprior المعدلة من أجل إيجاد مجموعات العناصر المتكررة وقواعد الترابط المقابلة لها، تعاني الخوارزمية السابقة من بطىء ناتج عن طريقة التوليد والاختبار المأخوذة من الخوارزمية الأساسية [12].

المرجع (Makanju A. et al.)، قام الباحث بتفريع خوارزمية FP-growth، باستخدام بنية معدلة عن البنية Map Reduce تسمى بـ Parent-Child MapReduce. من أجل توزيع عملية التنقيب عن العناصر المتكررة، والتي تعتمد على مبدأ فرق تسد بصورة ديناميكية ومتزامنة على العقد الحسابية ضمن التجمع. تم اختبار الخوارزمية المقترحة على عنقود حسابي مكون من 6 عقد حيث حققت الخوارزمية المقترحة زيادة في السرعة تقدر بثلاثة أضعاف سرعة خوارزمية FP-growth تفرعية باستخدام البنية Map Reduce عند التعامل مع قواعد البيانات الضخمة [15].

قام (Nikam P. V. et al.) بتقديم خوارزمية للتنقيب عن العناصر المتكررة بشكل تفرعي FiDoop، من خلال تطبيق نموذج برمجي موزع Map Reduce. يحقق توزيع البيانات على العقد بشكل أوتوماتيكي، وتحقيق التوازن في الأحمال بين عقد المعالجة. النظام المقدم في هذا العمل يستخدم نسخة من خوارزمية هجينة مكونة من خوارزمية Aprior وFP-growth وذلك من أجل دمج إيجابيات كل منهما والتقليل من البيانات المتبادلة بين العقد الحسابية. تم تحقيق تحسين في سرعة عملية التنقيب عن العناصر المتكررة [18].

في معظم الخوارزميات السابقة تم التغلب على مشكلة المساحة الذاكرية المطلوبة من أجل تخزين قاعدة البيانات الضخمة وذلك من خلال توزيعها على أكثر من عقدة حسابية. بالإضافة إلى تسريع عملية التنقيب عن العناصر المتكررة، على الرغم من زيادة تعقيد بعض الخوارزميات السابقة من خلال إضافة بنى مساعدة للبنى المستخدمة في الخوارزميات الأساسية. ولكن بعض الخوارزميات تعاني من عدم قابلية التوسع (Scalability)، وبعضها يعطي نتائج تقريبية (Aproximate frequent itemsets) بدلاً من النماذج المتكررة الحقيقية.

سوف يتناول بحثنا تطوير خوارزمية FP-growth من خوارزميات التنقيب الأساسية نتيجة المزايا السابقة الذكر والتي تتمتع بها[14] ، وذلك بهدف زيادة فعالية الخوارزمية. بالإضافة إلى زيادة سرعة الخوارزمية وتخفيض المساحة الذاكرية المطلوبة لتخزين تلك العناصر. وتمكين الخوارزمية المقترحة من التعامل مع قواعد البيانات الضخمة بمختلف أنواعها، وذلك من خلال استخدام إطار تخزين ومعالجة البيانات الموزعة هادوب. بما يحقق قابلية التوسع (Scalable) وسرعة تفوق التطبيق التقليدي لخوارزمية FP-growth على حاسب واحد فقط والعمل على موازنة الأحمال بين العقد الحسابية.

5. خوارزمية FP-growth الأساسية:

تعمل خوارزمية FP-growth الأساسية وفق الخطوات التالية:

1. المرور الأول لقاعدة البيانات لإيجاد 1-itemsets (مجموعة تتضمن كل عنصر بشكل منفرد) مع تكرارها، وتخزينها في اللائحة H.
2. الحصول على العناصر المتكررة والتي يكون تكرارها أكبر من عتبة يحددها المستخدم min_sup. وترتيبها تنازلياً بحسب قيمة تكرارها.
3. ترتيب عناصركل مداولة في قاعدة البيانات وفق اللائحة H.
4. المرور الثاني على قاعدة البيانات لبناء شجرة النموذج المتكرر FP tree، حيث يتم بناء الشجرة كما يلي:
   - إنشاء جذر الشجرة وتسميته null.
   - لكل سجل من سجلات قاعدة البيانات، يتم ترتيب العناصر المتكررة فيه بحسب الترتيب في اللائحة H.
   - يتم إنشاء فرع لكل سجل من سجلات قاعدة البيانات مع مراعاة ما يلي: إذا كانت الشجرة تتضمن عقدة لها الاسم نفسه للعنصر الحالي نزيد العداد الموجود فيها بمقدار واحد، وإلا يتم إنشاء عقدة جديدة يكون التكرار فيها مساوياً الواحد مع ربطها مع العقدة الأب لها.
5. التنقيب في شجرة FP tree: لكل نموذج بطول 1 (نموذج لاحق ابتدائي prefix pattern)، يتم بناء قاعدة النموذج الشرطي Conditional Pattern Base (قاعدة بيانات جزئية تشكل مسارات مرتبطة في شجرةFP الشرطية Conditional FP tree) لإيجاد العناصر المتكررة .يتم الحصول على النموذج الابتدائي مع النماذج المتكررة المولدة من شجرة FP الشرطية .[7] [8]

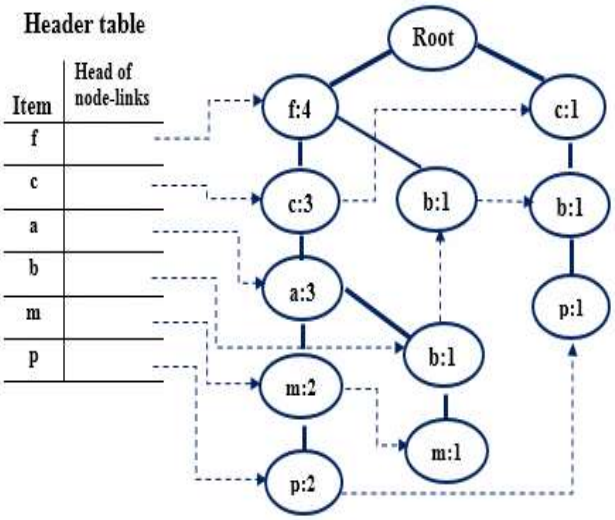
المثال التالي يوضح آلية عمل خوارزمية FP-growth حيث يبين الجدول 1 قاعدة المعطيات DB، مع افتراض أن عتبة التكرار هي min_sup=3، والعناصر المتكررة بعد ترتيبها بشكل تنازلي (ordered) Frequent Item وحذف العناصر ذات عدد مرات التكرار الأقل من min_sup، لترتيب المداولات وذلك بعد المرور الثاني والأخير على قاعدة المعطيات [7] [8].

الجدول 1: مداولات قاعدة المعطيات DB.

TID	Transactions	(ordered) Frequent Item
T100	f, a, c, d, g, i, m, p	f, c, a, m, p
T200	a, b, c, f, l, m, o	f, c, a, b, m
T300	b, f, h, j, o	f, b

T400	b, c, k, s, p	c, b, p
T500	a, f, c, e, l, p, m, n	f, c, a, m, p

أما الشكل 1 يبين جدول Header-table والذي يحوي العناصر المتكررة، وشجرة FP-Tree التي تمثل قاعدة المعطيات السابقة.

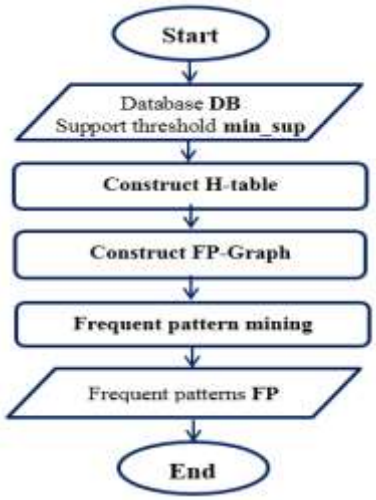


الشكل 1: شجرة FP-Tree.

6. الخوارزمية المقترحة:

نقترح تطوير خوارزمية FP-growth من خلال استبدال بنية الشجرة بالمخطط الموجه، حيث تتألف الخوارزمية المقترحة (**G**raph **F**requent **P**atteren GFP-growth) من ثلاث مراحل أساسية:

1. بناء جدول العناصر المتكررة H-table.
2. بناء المخطط الموجه FP-Graph.
3. التنقيب عن العناصر أو النماذج المتكررة. والشكل 2 يمثل المخطط التدفقي الذي يمثل الخطوات السابقة.



الشكل 2: المخطط التدفقي للخوارزمية المقترحة.

تم الاستعاضة عن شجرة FP-Tree بالمخطط الموجه FP-Graph. صمم FP-Graph من أجل تخفيض المساحة الذاكرية المطلوبة لتخزين العناصر المتكررة. تركز الخوارزمية المقترحة على تخفيض عدد عقد المخطط اللازمة لتخزين العناصر المتكررة لأول مجموعة (عنصر واحد -1 itemsets) ضمن مداولات قاعدة البيانات.

يمكن أن نُعرف المخطط (FP-Graph): بأنه عبارة عن مخطط موجه G يحوي مجموعة من العقد (Vertices)، V= {V_1,V_2,…….. Vn} ومجموعة من الحواف المرتبة (Edge)، E= {E_1, E_2, Ek}.

وإذا كان لدينا قاعدة البيانات يرمز لها بـ DB= {T_1,T_2,……,T_m} والتي تعتبر دخل للمخطط. يحوي المخطط عدد عقد مساوي للعناصر المتكررة لأول مجموعة (عنصر واحد 1-itemsets). كل حافة Eij بين عقدتين V_i وV_j تمثل جزء من مداولة (Subpath)، ومن أجل تخزين هذا الجزء عند وروده في أكثر من مداولة تم تزويد كل حافة بقائمة TransId-list وذلك لتخزين المداولات المشتركة بتلك الحافة. يتألف كل سطر في جدول العناصر المتكررة H-table من ثلاثة حقول، الأول مُعرف العنصر (itemId)، والثاني تكرار العنصر(Frequency count) ، أما الثالث مؤشر لعقدة هذا العنصر في المخطط(itemNodePointer).

كل عقدة في المخطط تتألف من حقلين، الأول مُعرف العنصر، والثاني عبارة عن قائمة (Parent-list) والتي تضم مؤشر آباء هذه العقدة(ParentNodePointer). ذكرنا سابقاً بأن الخوارزمية المقترحة تتألف من ثلاث مراحل، الفقرات التالية توضح المراحل السابقة الذكر بالتفصيل.

6.1 *بناء جدول العناصر المتكررة H-Table:*

بناء المخطط FP-Graph من قاعدة البيانات يتطلب ترتيب العناصر المتكررة بشكل تنازلي. يتم من خلال المسح أو المرور الأول لقاعدة البيانات بناء جدول العناصر المتكررة H-table. وهذه الخطوة مطابقة للخطوات الأولى في خوارزمية FP-growth الأساسية وبدون أي تعديل.

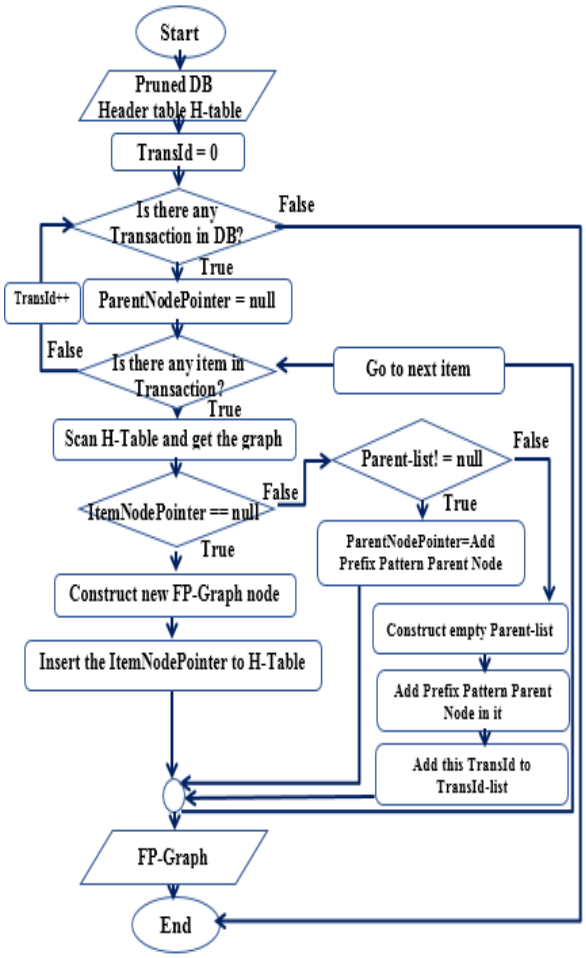
6.2 *بناء المخطط FP-Graph:*

يتم بناء المخطط الخاص بالعناصر المتكررة دون الحاجة إلى أي بنية مساعدة كما في الخوارزميات السابقة، حيث تقوم الخوارزمية المقترحة بمسح قاعدة المعطيات لمرة ثانية وأخيرة، من أجل ترتيب كل مداولة تنازلياً بالاعتماد على جدول H-table، وحذف كل العناصر غير المتكررة (تكرارها أقل من عتبة الدعم الدنيا). يستخدم رقم المداولة من أجل تعليم الحواف بين العقد، ومن ثم يتم ادخال المداولة إلى المخطط. يُشكل المخطط بنية مضغوطة (تمثيل كل عنصر بعقدة واحدة فقط، وحفظ جميع الأجزاء المتماثلة بين المداولات في نفس الحافة) وفعّالة لتخزين العناصر المتكررة لأول مجموعة -1) (itemset، كل سطر في جدول العناصر المتكررة H-table يحوي مؤشر لعقدة هذا العنصر في المخطط (itemNodePointer). يتم تخزين تكرار النموذج اللاحق باستخدام قائمة(Parent-list) بدلا من حفظ تكرار العنصر ضمن العقدة مما يؤدي لزيادة سرعة عملية التنقيب. وتُقسم عملية بناء المخطط إلى مرحلتين أساسيتين:

1. تهيئة العقد وربطها مع المؤشر itemNodePointer: الخطوات من 1 إلى 2.3.4 في الشيفرة الزائفة الموضحة لاحقاً.
2. حفظ المداولات في المخطط: الخطوات من 3 إلى 3.3 في الشيفرة الزائفة الموضحة لاحقاً أيضاً. والمخطط التدفقي الخاص بخوارزمية بناء المخطط موضح في الشكل 3.

بما أن المخطط لا يحوي عقدة جذر (Root) يتم العبور عبر العقد من خلال جدول H-table. أما البحث عن النموذج اللاحق (Prefix Pattern) يتطلب

فقط معرفة العقد الآباء لعقدة هذا النموذج ويتم تخزينها في قائمة-Parent) (list، لذلك لا يتم تخزين أبناء العقد في الخوارزمية المقترحة مما يحقق توفير بالزمن والمساحة التخزينية.



الشكل 3: المخطط التدفقي لبناء FP-Graph.

أما الشيفرة الزائفة لبناء المخطط موضحة في الجدول 2.

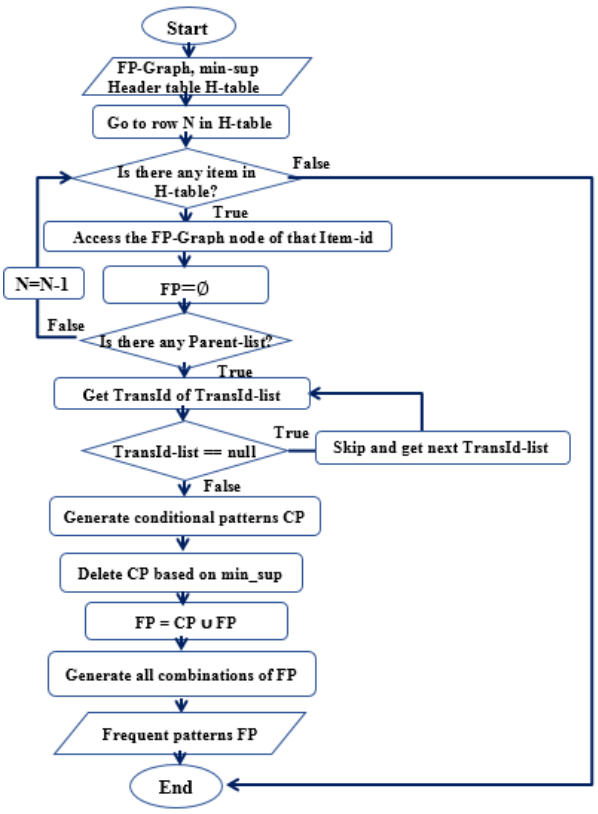
الجدول 2: الشيفرة الزائفة لبناء المخطط.

Algoithm1: Procedure of Construct FP-Graph.
Input: A transaction database **DB** and frequent item header table **H-table**. **Output**: **FP-Graph.**
Initialize TransId to 0. **For** each Transaction t ∈ DB. 2.1 Initialize ParentNodePointer = null. 2.2 transId++; 2.3 **For** each item i ∈ t. 2.3.1 ItemNodePointer = scan H-Table and get the graph pointer. 2.3.2 **IF** ItemNodePointer = = null. 2.3.3 Construct new FP-Graph node. 2.3.4 Insert the ItemNodePointer address into H-Table; /* Assume that the ItemNodePointer is filled at this stage. */ /* Adjust ParentNodePointer and TransId for tagging pattern-path with transId */ **IF** Parent-list! = null/* this current item-identifier node has parent */ ParentNodePointer=Add Prefix Pattern Parent Node. **Else** Construct empty Parent-list and then Add Prefix Pattern Parent Node in it. 3.3 Add this TransId to TransId-list.

6.3 التنقيب عن العناصر المتكررة: *Frequent itemset*

بعد بناء المخطط بطريقة مضغوطة، تتوقف عملية مسح قاعدة البيانات، ليتم البدء بالمرحلة الثالثة وهي التنقيب عن العناصر المتكررة، حيث يتم توليد العناصر المتكررة بدون إيجاد الشجرة الشرطية (كما في خوارزمية FP-growth) مما يؤدي إلى زيادة سرعة الخوارزمية المقترحة. يتم استخدام خوارزمية البحث من الأسفل إلى الأعلى من أجل عبور كل عقد المخطط. لإيجاد النموذج المتكرر من خلال الانطلاق من العقدة التي تمثل هذا النموذج، ثم يتم الانتقال إلى أباء هذه العقدة حتى نصل إلى عقدة مؤشر الأب ParentNodePointer لها يساوي Null. ومن أجل إيجاد النماذج المتكررة الشرطية لكل عنصر نستخدم اللائحة TransId-list. في خوارزمية التنقيب عن العناصر المتكررة، يتم الوصول إلى ParentNodePointer من القائمة Parent-list لكل عقد المخطط. ثم يتم استرجاع قيم TransId من القائمة TransId-list من أجل كل ParentNodePointer لتوليد النماذج الشرطية. بعد توليد النماذج الشرطية CP يتم حساب قيمة الدعم لها من أجل القيام بحذف النماذج الشرطية CP والتي لاتحقق عتبة الدعم الدنيا min-sup. ليتم بعدها إيجاد مجموعات النماذج المتكررة المتاحة النهائية.

والمخطط التدفقي المعبر عن خوارزمية التنقيب عن العناصر المتكررة موضح في الشكل 4.



الشكل 4: المخطط التدفقي للتنقيب عن العناصر المتكررة Frequent patterns.

والشيفرة الزائفة لخوارزمية التنقيب عن العناصر المتكررة موضحة في الجدول 3:

الجدول 3: الشيفرة الزائفة للتنقيب عن العناصر المتكررة.

Algorithm2: Procedure of Frequent itemset mining.
Input: FP-Graph, Support threshold min_sup and frequent item header table H-table. Output: Frequent patterns FP.
For (each bottom up item from H-table). Access the FP-Graph node of that Item-id. 1.2 FP = {Ø}. 1.3 For (each Parent-list). 1.3.1 Get TransId of TransId-list. 1.3.2 IF (TransId-list is empty). 1.3.3 Skip and get next TransId-list. 1.3.4 For (each TransId in TransId-list). 1.3.5 Generate conditional patterns CP. 1.3.6 Delete CP based on min_sup. 1.3.7 FP = CP ∪ FP. 1.3.8 Generate all combinations of FP.

مثال توضيحي: سنقوم بتطبيق الخوارزمية المقترحة على قاعدة المعطيات الموضحة سابقاً في الجدول1 ، مع افتراض أن min_sup=3. بعد مسح قاعدة البيانات، يتم توليد جدول H-table الذي يحوي العناصر المتكررة مرتبه تنازلياً، حيث يتم حذف كل العناصر الغير متكررة (التي يكون تكرارها أقل من min_sup=3) من الجدول. في الخوارزمية 1 يتم بناء المخطط FP-Graph من خلال مسح قاعدة المعطيات مرة ثانية وأخيرة، وترتيب كل مداولة تنازلياً وحذف العناصر غير المتكررة منها بالاعتماد على جدول H-table كما يوضح الجدول1 -العمود Frequent Item (ordered). بعد تطبيق قاعدة البيانات الموضحة في الجدول 1 على ال خوارزمية1 يتم بناء FP-Graph كما هو موضح في الشكل 5.

كل المداولات يتم تمثيلها من خلال Parent-list والتي تحوي مؤشرات للعقد الآباء لهذه العقدة ParentNodePointer (حيث تمثل عقدة عنصر ما Ii عقدة أب لعنصر آخر Ij إذا كان تكراره ضمن H-table يساوي أو أكبر من العنصر Ij ويتواجدان ضمن نفس المداولة)، بالإضافة إلى TransId-list التي تحوي أرقام المداولات والتي تستخدم من أجل تعليم (Tagging) الحواف بين عقد المخطط. عدد العقد في FP-Graph يساوي عدد العناصر المتكررة (6 عناصر) والمكونة من عنصر واحد -1 (frequent itemsets).

في المثال السابق يحوي المخطط ست عقد كما هو موضح في الشكل 5. المداولة الأولى (f, c, a, m, p) ذات المُعرف أو الرقم T100، تخزن ضمن المخطط، حيث إن العقدة p تشير إلى العقدة الأب mويتم تعليم الحافة بينهما بـ T100، العقدة m تشير إلى العقدة الأب a ويتم تعليم الحافة بينهما بـ T100، العقدة a تشير إلى العقدة الأب c ويتم تعليم الحافة بينهما بـ T100، العقدة c تشير إلى العقدة الأب f ويتم تعليم الحافة بينهما بـ T100، أما مؤشر ParentNodePointer الخاص بالعقدة f يساوي null. المداولة رقم T300 (f, b) العقدة b تشير إلى العقدة الأب f ويتم تعليم الحافة بينهما بـ T300، أما مؤشر ParentNodePointer الخاص بالعقدة f يساوي null. ويمكن تعميم ما سبق على كل المداولات. من خلال المخطط، يمكن أن نجد تكرار أي عنصرين متكررين، مثلاً {p m} من خلال الحافة المشتركة بين عقدتيهما نجد أن اجتماع هذان العنصران في المداولات [T100,T500]. العنصران {a b} نلاحظ من المخطط أيضاً أن العقدة b تملك مؤشر ParentNodePointer مع العقدة a وقيمة الحافة المشتركة بينهما [T200].

يتم تطبيق الخوارزمية 2 على المخطط الذي تم بناؤه في المرحلة السابقة للتنقيب عن العناصر المتكررة الموضحة في الجدول 3.

تبدأ عملية التنقيب من العنصر الموجود أسفل الجدول H-table، وبتطبيق min_sup=3، يتم الانطلاق من سطر العنصر p إلى المخطط FP-Graph من أجل إيجاد كل المداولات التي تحوي العنصر p ودون الحاجة لمسح قاعدة البيانات. تمتلك العقدة p مؤشرين ParentNodePointer يشيران إلى العقدتين m، b وبقيمة حافة [T100,T500]، [T400] على الترتيب. من أجل المداولة T100 يتم الانتقال من العقدة p إلى العقدة m ثم a ثم c ومن ثم f. والمداولة T500 يتم الانتقال من العقدة p إلى العقدة m ثم a ثم c ومن ثم f أيضاً.

أما المداولة T400 يتم الانتقال من العقدة p إلى العقدة b ثم c. وبالتالي النماذج الشرطية الخاصة بالعقدة: p {b, c}،{m, a, c, f}، {m, a, c, f} وبنفس الطريقة يتم استنتاج النماذج الشرطية لباقي العناصر. بعد ذلك يتم تطبيق عتبة الدعم الدنيا من أجل حذف النماذج الشرطية غير المتكررة (العمود conditional patterns based on min_sup من الجدول 4)، ليتم بعدها توليد النماذج المتكررة بشكلها النهائي (العمود Frequent pattern من الجدول 4) من خلال إيجاد كل المجموعات الممكنة من النماذج الشرطية المتكررة.

الجدول 4: النماذج المتكررةFrequent pattern .

Conditional patterns	conditional patterns based on min_sup	Frequent pattern
p:{m, a, c, f}|{b, c} |{ m, a, c, f }	p:{c}	{p}, {p ,c}
m:{ a, c, f }| { b, a, c, f}| { a, c, f }	m:{a, c, f}	{m}, {m, a}, {m, c }, { m, f }, {m, a, c}, {m, a, f}, {m, c, f}, {m, a, c, f}
b:{c}|{f}|{ a, c, f}	b:{ }	{b}
a:{c, f}|{ c, f }| {c, f }	a:{c, f}	{a},{a, c, f}, {a, c}, {a, f}
c:{f}|{f}|{f}	c:{f}	{c}, {c, f}
f:{ }	f:{ }	{ f }

6.4 *دراسة فعالية الخوارزمية المقترحة:*

إن نسبة كبيرة من زمن التنفيذ خوارزمية FP-growth الأساسية يستهلك في بناء شجرة FP-Tree، والتنقيب في شجرة FP-growth الشرطية. ولتقليص هذا الزمن وخاصة الزمن المستهلك من أجل التنقيب عن العناصر المتكررة وبناء الشجرة تم اقتراح التطوير في الفقرة السابقة بالاعتماد على المخطط الموجه كما تم شرحه.

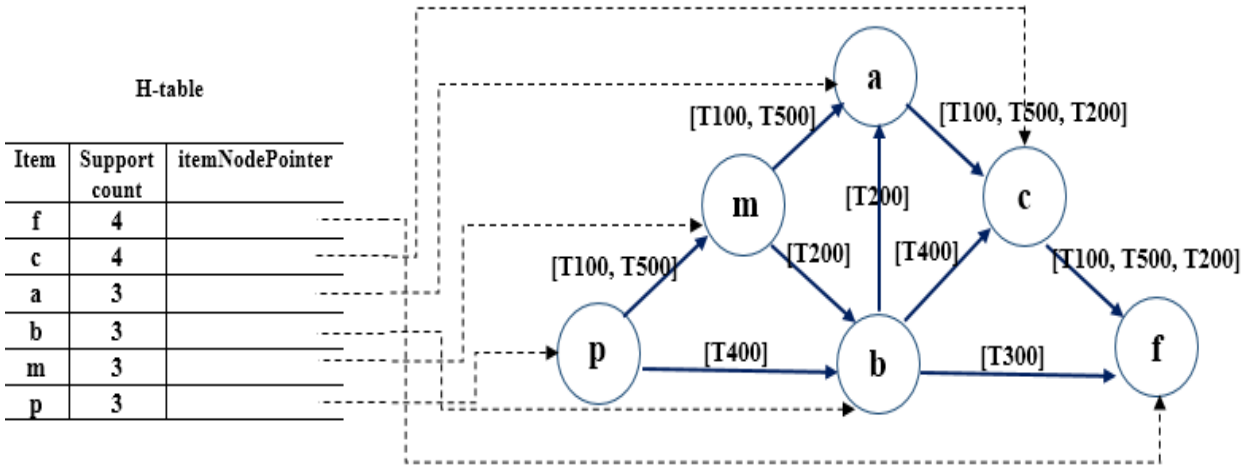
إذا كان لدينا مجموعة I والتي تحوي n عنصر، وإذا كان لدينا m مداولة ضمن قاعدة البيانات DB، حيث إن كل مداولة تحوي l عنصر.

أفضل حالة: درجة تعقيد الخوارزمية من أجل التنقيب عن العناصر المتكررة هو$O(n^2)$ + O (n log n) + O (m + n*l). فمن أجل توليد جدول H-table، فإننا نقوم بإنشاء n سجل في جدول H-table، وذلك من خلال قراءة l عنصر في m مداولة وبالتالي فإن درجة التعقيد O (m + n*l). يتطلب بناء FP-Graph توليد n عقدة، تمتلك كل عقدة عدد من الحواف والمؤشرات تساوي عدد العقد الآباء لتلك العقدة. أفضل حالة هي أن تكون كل مداولة تحوي كل عناصر قاعدة البيانات، وبالتالي كل المداولات تمثل من خلال نفس المسار ضمن المخطط الموجه، أي يتم بناء المسار من خلال أول مداولة ثم يتم تعديل لائحة TransId-list فقط في باقي المداولات. وبالتالي درجة التعقيد O (n log n). أما عند التنقيب عن النماذج المتكررة، فإذا كان لدينا q نموذج بطول e عنصر، فإن درجة تعقيد إيجاد العناصر المتكررة هي $O(n^2)$ =O(q*e).

أسوء حالة: إن درجة تعقيد الخوارزمية المقترحة بالنسبة للزمن للتنقيب عن العناصر المتكررة هو$O(n^2)$ + O (n^2) + O (m + n*l).

في هذه الحالة الاختلاف فقط في مرحلة بناء المخطط الموجه، يتم تمثيل كل مداولة بمسار مختلف عن المداولات الأخرى. وبالتالي استهلاك زمن إضافي لزيارة كل عقدة واضافة مؤشر للعقدة الأب الخاص بها وبالتالي درجة التعقيد O (n(n-1)) = O (n^2).

ولبرهان فعالية الخوارزمية المقترحة رياضياً ومقارنتها بخوارزمية FP-Growth فمن ناحية **المساحة الذاكرية:** فإن كل سجل في جدول H-table يتطلب 10 بايت، 2 بايت لمُعرف العنصر، و4 بايت لتكرار العنصر، و4 بايت لمؤشر عقدة هذا العنصر في المخطط. كل عقدة في المخطط كما توضح العلاقة1 تحتاج لـ 2 بايت من أجل معرف العنصر (itemId)، 4 بايت من



الشكل 5: مخطط FP-Graph لقاعدة البيانات السابقة.

أجل Parent-list، 4 بايت من أجل TransId-list، حيث N: تمثل عدد العقد الآباء لهذه العقدة.

$$Space\ complexity(vertix) = 2 + 4N + 4N = 2 + 8N \quad (1)$$

وبالتالي لتوضيح فعالية البنية المقترحة FP-Graph ومدى ضغطها لقاعدة المعطيات، سوف نقوم بحساب المساحة التخزينية المطلوبة للمثال السابق، العدد الكلي لعقد المخطط هو 6 عقد والمساحة التخزينية المطلوبة تكون على الشكل التالي:

المساحة المطلوبة لجدول H-table: 10*6= 60 بايت.

المساحة المطلوبة للمخطط FP-Graph: كل عقدة تتطلب مساحة تخزينية تختلف باختلاف عدد العقد الآباء لتلك العقدة، ويساوي 144 بايت كما هو موضح تالياً: فمثلاً العقدة p تحوي مؤشرين ParentNodePointer في اللائحة Parent-list الأول يشير إلى العقدة m والثاني يشير إلى العقدة b وبالتالي فإن N=2 بالنسبة للعقدة p. وبالتالي المساحة المطلوبة لكل عقدة تكون على الشكل التالي: العقدة p: 2+ (2*8) =18 بايت. العقدة m: 2+ (2*8) =18 بايت. العقدة b: 2+ (3*8) =26 بايت. العقدة a: 2+ (1*8) =10 بايت. العقدة c: 2+ (1*8) =10 بايت. العقدة f: 2 = 2بايت. المساحة الكلية المطلوبة لتخزين العقد = 84 بايت. المساحة الكلية المطلوبة لتخزين المخطط الخاص بالعناصر المتكررة FP-Graph = 60+84 =144 بايت.

بالمقارنة بين الخوارزمية المقترحة وخوارزمية FP-growth من حيث عدد العقد اللازمة لتخزين قاعدة المعطيات في المثال السابق والمساحة التخزينية، يتألف FP-Graph في الخوارزمية المقترحة من 6 عقد لتخزين قاعدة المعطيات في المثال السابق مقابل 12 عقدة في شجرة FP-Tree. أما المساحة التخزينية المطلوبة لتخزين من أجل FP-Graph 144 بايت مقابل 292 بايت وفق المرجع [7] من أجل شجرة FP-Tree. أي توفير بمقدار 50% من حيث عدد العقد والمساحة التخزينية المطلوبة.

أما من حيث **الزمن:** يعتمد الزمن الذي تستغرقه الخوارزمية المقترحة على ثلاثة أزمنة رئيسية كما هو موضح في العلاقة2:

$$Time\ complexity = n * T1 + m * T2 + r * T3 \quad (2)$$

T1: يمثل الزمن اللازم لتوليد سجل واحد في جدول H-table (يتضمن زمن قراءة كل عنصر في مداولات قاعدة المعطيات لتوليد السجل الخاص به، أو تعديل قيمة الحقل Support count في حال وجود سجل سابق له). n: العدد الكلي للسجلات. T2: يمثل الزمن اللازم لتوليد عقدة واحدة ضمن المخطط. m: العدد الكلي للعقد. T3: يمثل الزمن اللازم لقراءة عقدة واقعة في مسار النموذج المتكرر الخاص بكل عنصر موجود في جدول H-table (التنقيب عن النماذج المتكررة الجدول3). r: عدد العقد المقروءة. لتوضيح فعالية الخوارزمية المقترحة، سوف نقوم بدراسة الزمن المطلوب للمثال السابق: الزمن اللازم لبناء جدول H-table من خلال الخوارزمية المقترحة يساوي T1 *16 (يتضمن قراءة 33 عنصر، وتوليد 16 سجل للعناصر المتكررة وغير المتكررة) وهو نفس الزمن اللازم لبناء الجدول في خوارزمية FP-Growth الفقرة (6-1). أما الزمن الكلي لبناء المخطط يساوي6*T2 (6 عقد) مقابل 12*T2 لبناء شجرة FP-tree والتي تحوي 12 عقدة المرجع [7]، وبفرض التنفيذ على نفس الجهاز. من أجل التنقيب عن مجموعة العناصر المتكررة الخاصة بالعقدة p في الخوارزمية المقترحة سيتم قراءة 7 عقد والزمن اللازم لإيجادها 7*T3 مقابل 10*T3 في خوارزمية FP-Growth. أما زمن توليد النماذج المتكررة الخاصة بالعقدة m= 8*T3 مقابل 11*T3 في خوارزمية FP-Growth. العقدة b= 6*T3 مقابل 11*T3 في خوارزمية FP-Growth. العقدة a= 3*T3 مقابل 4*T3 في خوارزمية FP-Growth. العقدة c= 2*T3 مقابل 5*T3 في خوارزمية FP-Growth. العقدة f= 1*T3 مقابل 2*T3 في خوارزمية FP-Growth. وبالتالي الزمن الكلي لتنفيذ المثال السابق في الخوارزمية المقترحة GFP-Growth = 27*T3+6*T2+16*T1، أما خوارزمية FP-Growth= 43*T3+12*T2+16*T1. بالمقارنة بين زمن تنفيذ الخوارزميتان نجد أن الخوارزمية المقترحة أسرع بمقدار 60% تقريباً من وخوارزمية FP-Growth في هذا المثال.

7. تطبيق الخوارزمية المقترحة تفرعياً:

عندما تكون قاعدة المعطيات كبيرة جداً، فإن الزمن اللازم لعملية التنقيب بالإضافة إلى المساحة التخزينية يزداد بصورة كبيرة، ممايؤدي إلى فشل عملية التنقيب في معظم الخوارزميات التسلسلية التقليدية.

لذلك تم اقتراح خوارزمية PGFP-growth، وهي عبارة عن خوارزمية تفرعية معتمدة على المخطط الموجه في بيئة حوسبة سحابية تفرعية. تهدف لحلّ المشاكل التي تواجه خوارزمية FP-growth الأساسية. تستخدم المخطط الموجه من أجل توفير المساحة التخزينية اللازمة لتخزين قاعدة المعطيات، والتخلص من توليد شجرة FP-Tree الشرطية مما يعني زيادة سرعة عملية التنقيب. وبنفس الوقت تعتمد على إطار هادوب لمعالجة وموازنة البيانات الضخمة (الأحمال) بين العقد الحسابية بشكل متزامن وفعّال وموزع من خلال البنية Map Reduce.

يعتبر هادوب برنامج مفتوح المصدر بإطار تخزين موزع لمجموعات كبيرة من البيانات في تجمعات حاسوبية (Computer Clusters). طوّر من قبل مؤسسة Apache وطبّق في العديد من الشركات مثل Facebook, Yahoo, Amazon, IBM والكثير من المواقع الكبرى.

يتميز بالعديد من الخصائص مثل انخفاض الكلفة المادية، حيث يمكننا بناء هادوب انطلاقاً من جهاز واحد أو أكثر. مع إمكانية إضافة أجهزة أخرى عند الحاجة لمساحة تخزينية إضافية دون أن يؤثر ذلك على النظام وبالتالي تحقيق خاصية قابلية التوسع (Scalability). حلّ مشكلة فشل أحد الأجهزة أثناء عملية المعالجة، من خلال تكرار نسخ البيانات في عدد من الأجهزة، ففي حال تعطل أي جهاز سيقوم هادوب تلقائياً بتكرار البيانات التي كانت في الجهاز المعطل أي تحقيق خاصية الاعتمادية (Reliability). والعمل على معالجة البيانات بشكل متزامن وفعّال من خلال أحد أهم مكوناته البنية Map Reduce من أجل زيادة السرعة، موازنة الأحمال بين العناقيد، بالإضافة إلى الإنتاجية العالية (High Throughput) والتي تمثل كمية العمل المنجز في واحدة الزمن [2][5].

الفكرة الأساسية في هادوب هي نقل البرامج إلى مكان البيانات على عكس الطرق التقليدية والتي يتم فيها نقل البيانات إلى البرامج، والهدف الرئيسي في اتباع هذا النهج هو أن نقل البيانات الضخمة يستغرق وقت طويل، لذلك تتم معالجتها في مكانها.

أي أن معظم عمليات المعالجة تجري في محطات أو عقد البيانات (Data Node) حيث توجد البيانات مما يقلل حركة المرور في الشبكة.

يتكون هادوب من ثلاث مكونات أساسية:

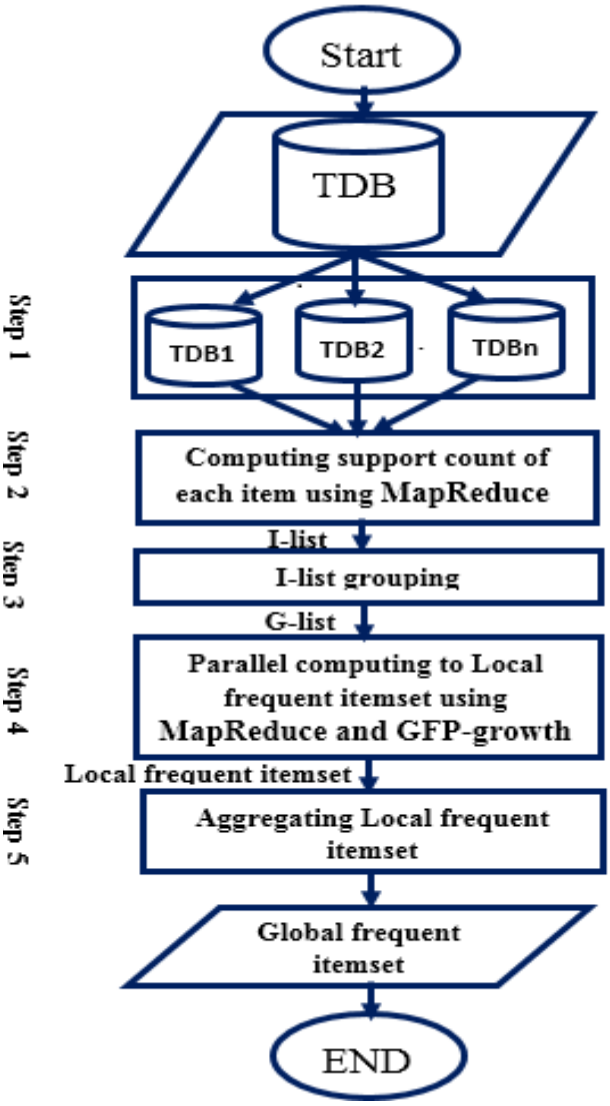
1. HDFS (**H**adoop **D**istributed **F**ile **S**ystem): وهو نظام الملفات المستخدم لتخزين البيانات.
2. MapReduce: وهو عبارة عن بيئة برمجية تستخدم المعالجة المتوازية في معالجة البيانات.
3. YARN (**Y**et **A**nother **R**esource **N**egotiator): وهو الجزء المسؤول عن إدارة التجمعات (Cluster).

الاعتماد الأساسي للهادوب في عملية المعالجة على تقنية Map وتقنية Reduce حيث تتم على ثلاث:

1. تقوم map بتقسيم البيانات إلى أجزاء (مفتاح Key/ قيمة Value).
2. ترسل مخرجات map إلى مرحلة Shuffle لترتيبها.

3. تقوم reduce بأخذ مخرجات المرحلة السابقة كمدخلات لتجميعها [20].

لتطبيق خوارزمية PGFP-growth على إطار هادوب التفرعي، نتبع الخطوات التسلسلية التالية الموضحة في الشكل 6:



الشكل 6: المخطط التدفقي للخوارزمية التفرعية المقترحة PGFP-growth.

1. **تقسيم قاعدة المعطيات:** في هذه المرحلة يتم تجزئة قاعدة المعطيات DB إلى أقسام متساوية يسند كل منها إلى عقدة حسابية، يحوي كل قسم عدد محدد من المداولات. نحتاج إلى تحميل قاعدة البيانات على نظام الملفات الخاص بهادوب، ليقوم هادوب تلقائياً بتقسيم قاعدة البيانات على العقد الحسابية بشكل متساوي.

2. **حساب تكرار العناصر:** يتم حساب تكرار كل عنصر من عناصر قاعدة المعطيات من خلال البنية map التي تُشكل قائمة تحوي الأزواج (مفتاح/ قيمة = T_i)، حيث $T_i \subseteq DB$ هي عبارة عن مداولة محتواه ضمن قاعدة المعطيات. ومن أجل كل عنصر $I_j \subseteq Ti$ تقوم البنية map بتوليد الزوج <Key=Ij, value=1>، ثم تنتقل مخرجات البنية map إلى المرحلة shuffle التي تقوم بترتيبها حسب المفتاح (العنصر)، ومنها إلى البنية reduce والتي تعمل على تجميع نتائج المرحلة السابقة، أي تقوم بجمع القيم الخاصة بكل عنصر أي <Key=Ij, sum(value)>. وخرج هذه المرحلة القائمة I-List، وهي عبارة عن قائمة تحوي العناصر المتكررة مرتبة تنازلياً. الشكل 7 يوضح تطبيق البنية mapreduce على قاعدة المعطيات (المداولة الأولى والثانية للسهولة) في المثال السابق.

3. **تجميع العناصر المتكررة:** في البحث [14] يتم تقسيم القائمة I-List والتي تحوي العناصر المتكررة مرتبة تنازلياً إلى Q مجموعة تحوي كل منها عدد متساوي من العناصر، تسمى بـ G-List يتم تميزها من خلال معرف خاص G-id. لايأخذ البحث السابق مفهوم ترابط العناصر أثناء عملية التجميع ضمن نفس المجموعة بعين الأعتبار، مما يؤدي الى زيادة حجم البيانات المتبادلة بين العقد الحسابية، بالإضافة الى مشكلة عدم التوازن في الأحمال بين العقد الحسابية (Imbalanced load). لحلّ المشاكل السابقة تم في البحث [23] اقتراح تجميع المداولات عن طريق إيجاد درجة الترابط بينها لتجميعها ضمن تجمع واحد وبمساعدة خوارزمية مطورة عن خوارزمية k-means تسمى k-means++، وعلى الرغم من فعالية الحلّ المقترح في هذا البحث الا أن عمليات المقارنة تستغرق زمن كبير عندما تكون قاعدة البيانات ضخمة. أما في الخوارزمية المقترحة يتم تجميع العناصر المتشابهة إلى Q مجموعة تحوي كل منها عدد من العناصرالمتشابهة، تسمى بـ G-List يتم تميزها من خلال معرف خاص G-id، عملية التجميع في الخوارزمية المقترحة تتم بالاعتماد على درجة الترابط Correlation Degree بين العناصر المتشابهة بهدف تسريع عملية التنقيب. درجة الترابط هي عبارة عن مقياس رياضي يحدد مدى التطابق بين مجموعتان من البيانات. يستخدم مقياس Jaccard similarity من أجل إيجاد مدى الترابط والاختلاف بين البيانات. تتراوح قيمة مقياس Jaccard بين القيمة 0 و1، ومع اقتراب القيمة إلى 1 يزداد التطابق بين البيانات [17] [23]. ويعطى بالعلاقة التالية:

$$Jaccard(I_i,I_j)=\frac{Ii \cap Ij}{Ii \cup Ij} \qquad (3)$$

حيث يتم تجميع العناصر المتشابهة ضمن مجموعة واحدة تسمى بـ G-List يتم تميزها من خلال معرف خاص G-id. تنفذ هذه العملية من خلال عقدة حسابية واحدة.

4. **تطبيق خوارزمية GFP-growth بالشكل التفرعي من أجل توليد العناصر المتكررة الخاصة بكل عقدة حسابية:** في الخوارزمية المقترحة يقوم التابع map بترتيب كل مداولة ترتيباً تنازلياً من اليسار الى اليمين كما يوضح الجدول 5 (العمود الثاني)، ومن ثم يتم ربط تلك المداولات مع المجموعات G_i-List. عن طريق إيجاد درجة الترابط أيضاً. الخطوة التالية البدء بعمل التابع reduce حيث يقوم التابع reduce ببناء المخطط FP-Graph الخاص بكل مجموعة متشابهة من المداولات وفق الخطوات الموضحة في الفقرة (6-2)، ومن ثم القيام بالتنقيب ضمن المخطط من أجل إيجاد قواعد الترابط الخاصة بتلك المداولات. تعتبر هذه المرحلة من المراحل الحرجة والأكثر تأثيراً على الزمن الكلي لعملية التنقيب[14] ، كما أن نتائج هذه المرحلة تحدد فعالية عملية التنقيب. والجدول 5 يوضح آلية عمل الخوارزمية المقترحة من خلال التابع map.

الجدول 5: عمل التابع map في الخوارزمية PGFP-growth.

Map inputs (transactions) Key="":value	(ordered) Frequent Item	Map outputs (transactions) Key: value
T100:	f c a m p	p: m a c f

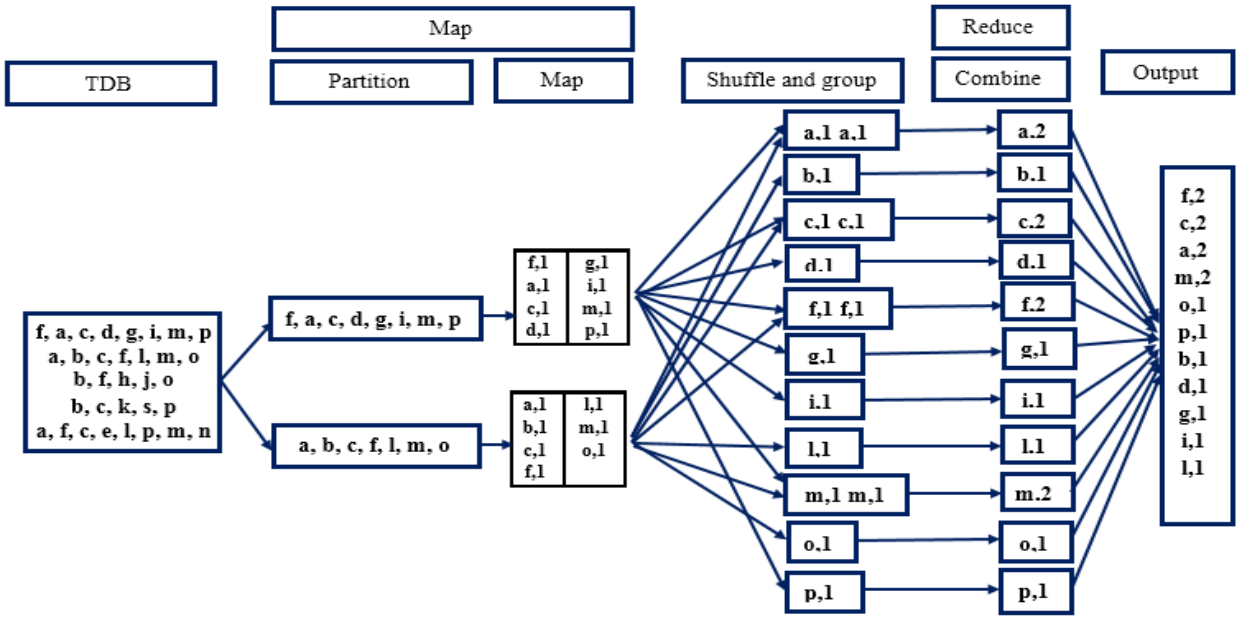
f a c d g I m p		m: a c f a: c f c:f
T200: a b c f l m o	f c a b m	m: b a c f b: a c f a: c f c: f
T300: b f h j o	f b	b: f
T400: b c k s p	c b p	p: b c b: c
T500: a f c e l p m n	f c a m p	p: m a c f m: a c f a: c f c:f

أما الجدول 6 يوضح آلية عمل الخوارزمية المقترحة من خلال التابع reduce وفق عتبة الدعم الدنيا (min_sup=3) من أجل حذف النماذج الشرطية غير المتكررة.

الجدول 6: عمل التابع Reduce في الخوارزمية PGFP-growth.

Reduce inputs (transactions) Key: value	Reduce outputs Frequent conditional patterns based on min_sup
p: { m a c f/ b c / m a c f }	{(c:3)}|p
m: {a c f/ b a c f / a c f }	{(a:3, c:3, f:3)}| m
b: {a c f/ f/ c}	{}|b
a :{c f/ c f/ c f}	{(c:3, f:3)}|a
c : {f/ f/ f}	{(f:3)}|c

5. **إيجاد الخرج النهائي**: من خلال تجميع نتائج كل العقد الحسابية الناتجة في المرحلة السابقة، يتم إيجاد مجموعات العناصر المتكررة بشكلها النهائي.



الشكل 7: تطبيق البنية MapReduce على قاعدة المعطيات السابقة (المداولة الأولى والثانية فقط للتبسيط).

8. النتائج التجريبية:

تم بناء خوارزمية PGFP-growth المقترحة بلغة الـ Java وبيئة Hadoop، وتم تطبيق هذه الخوارزمية والحصول على النتائج التجريبية من خلال عنقود هادوب مكون من خمس عقد، عقدة أب (Master) وأربع عقد أبناء (Slaves). كل العقد تمتلك نفس البنية البرمجية والمادية، نظام التشغيل Ubuntu 16.4 LTS-64Bit، ومعالج Intel®Core™i3 2.30GHz وذاكرة RAM 6GB. واصدار Hadoop-2.8.0، JDK1.8.0_181. سيتم مقارنة الخوارزمية المقدمة مع PFP-growth [13] وخوارزمية FP-growth [7] الأساسية، لإثبات فعالية هذه الخوارزمية عند التعامل مع قوعد البيانات الضخمة، وتوفير الزمن اللازم لعملية التنقيب والمساحة التخزينية المطلوبة، وتخفيض التعقيد الحسابي والاتصال بين عقد التجمع، وذلك عند

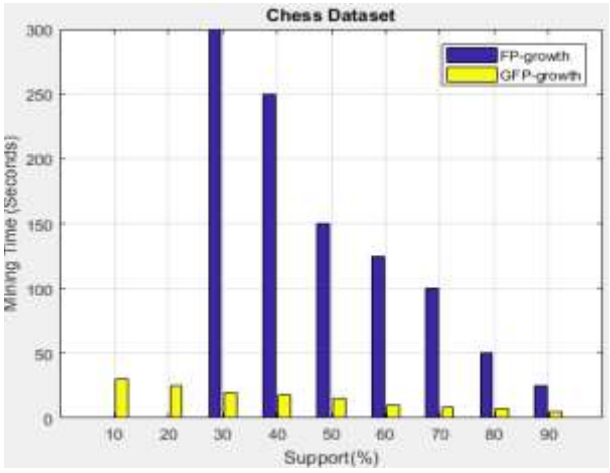
تطبيق عتبات min-sup مختلفة. استخدمت مجموعة من قواعد معطيات قياسية في أبحاث سابقة، تم تحميلها من موقع FIMI'04 (FIMI 2004) [25]، لإجراء الاختبارات. الجدول 7 يبين أهم خصائص قواعد المعطيات المستخدمة في اختبار الخوارزميات وهي على الترتيب عدد المداولات، متوسط طول المداولة، عدد العناصر، الحجم.

الجدول 7: خصائص قواعد المعطيات المستخدمة.

Dataset	Trans	Avg. Length	Items	Size (MB)
Chess	3196	37	75	0.34
Accident	340183	34	468	14.7
T40I10D100K	100000	10	942	30.25
Kosarak	990002	8	41270	40.5

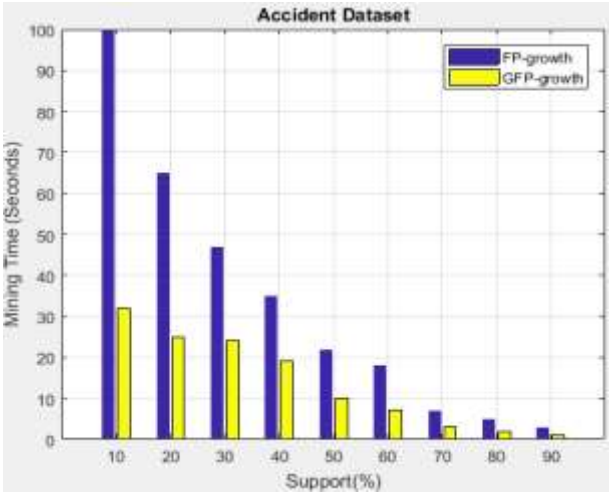
يوجد نوعان من قواعد المعطيات: متفرقة (Sparse) أو كثيفة (Dense)، وهو مصطلح يستخدم في قواعد البيانات متعددة الأبعاد لتحديد نسبة الخلايا المستخدمة وغير المستخدمة ضمن جداول قاعدة البيانات، وكثافة قاعدة المعطيات تُعرف على أنها متوسط طول المداولة مقسوم على العدد الكلي للعناصر ضمن قاعدة المعطيات، عملياً يتم اعتبار قاعدة المعطيات كثيفة إذا كانت نسبة الخلايا المستخدمة تزيد عن 10% [25].

في البداية سوف يتم مقارنة النتائج التجريبية الخاصة بالخوارزمية المقترحة المحسنة GFP-growth مع FP-growth الأساسية وعلى عقدة حسابية واحدة لإثبات مدى فعالية الخوارزمية المحسنة، باستخدام قاعدة المعطيات الكثيفة Chess وبتطبيق عتبات دعم مختلفة min_sup من 10% إلى 90%.



الشكل 8: زمن التنفيذ من أجل عتبات دعم مختلفة **(Chess).**

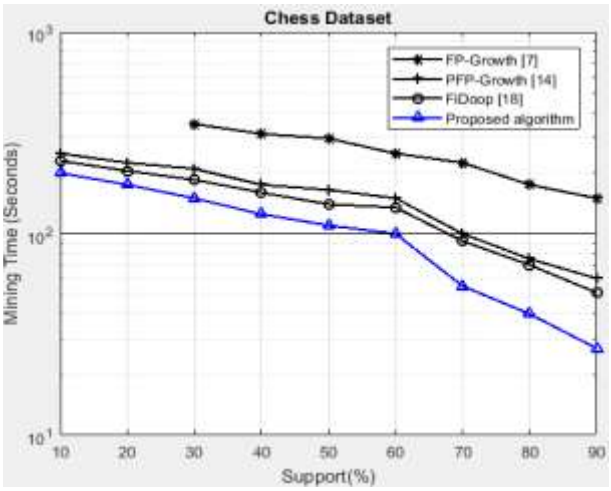
من الشكل 8 نجد أن الخوارزمية المقترحة تحقق زمن تنفيذ أسرع من خوارزمية FP-growth الأساسية (بنسبة تزيد عن 10 أضعاف عند بعض العتبات)، مع ملاحظة عدم قدرة خوارزمية FP-growth الأساسية التعامل مع قاعدة المعطيات Chess من النوع الكثيف (Dense)، عند عتبات الدعم المنخفضة مثل 10% و20% كما هو موضح بالشكل.



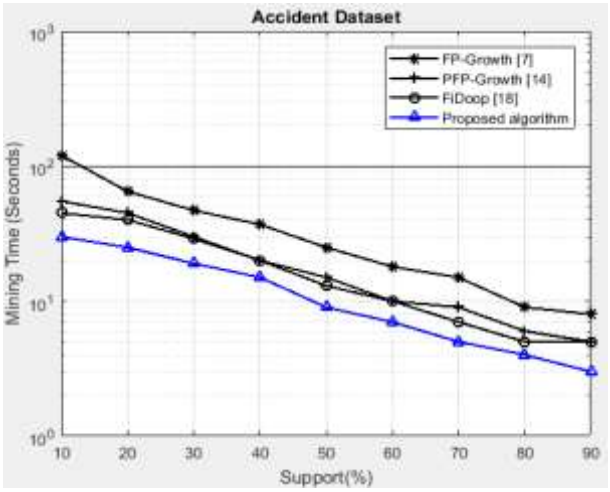
الشكل 9: زمن التنفيذ من أجل عتبات دعم مختلفة **(Accident).**

الشكل 9 يبين أن الخوارزمية المقترحة أيضاً تحقق زمن تنفيذ أقل من خوارزمية FP-growth الأساسية، عند التعامل مع قاعدة المعطيات Accidents من النوع المتفرق (Sparse).

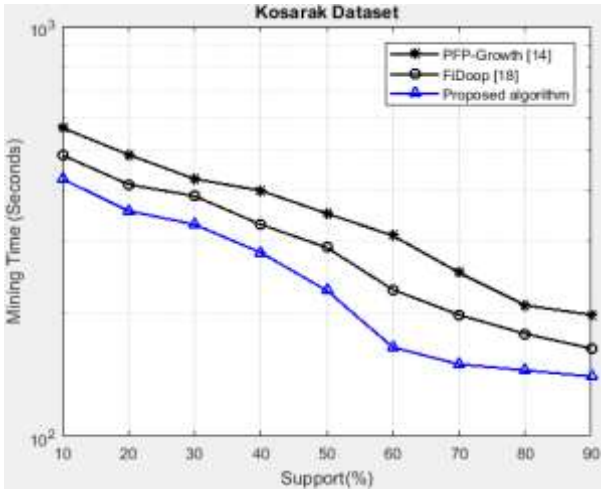
كما تمت مقارنة الخوارزمية التفرعية المقترحة PGFP-growth مع FP-growth [7]، وخوارزمية مع PFP-growth [14] وخوارزمية FiDoop [18] (من أجل مقارنة الخوارزمية المقترحة مع نوع آخر من الخوارزميات الأساسية للتنقيب على اعتبار أن الخوارزمية المقترحة في المرجع[18] هي خوارزمية هجينة مكونة من خوارزمية Aprior وFP-growth). كما هو مبين في الأشكال 10، 11، 12، 13.



الشكل 10: زمن التنفيذ من أجل عتبات دعم مختلفة **(Chess).**



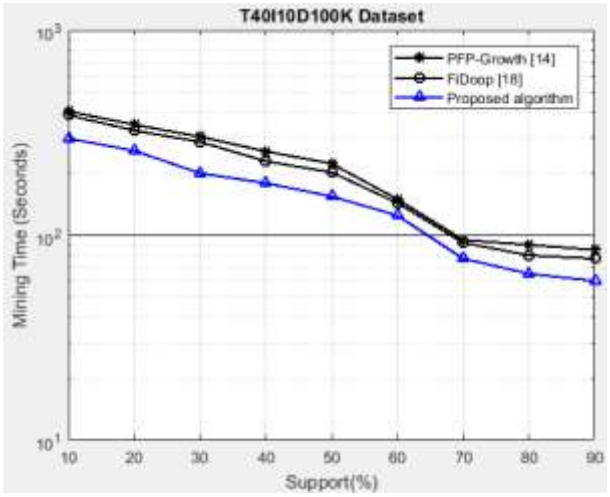
الشكل 11: زمن التنفيذ من أجل عتبات دعم مختلفة **(Accident).**



الشكل 13: زمن التنفيذ من أجل عتبات دعم مختلفة **(Kosarak).**

إن الخوارزمية المقترحة تحتاج زمن تنفيذ أقل للتنقيب عن العناصر المتكررة الخاصة بقاعدة المعطيات Chess وقاعدة المعطيات Accident وبعتبات دعم مختلفة (90%-10%) من الخوارزمية مع PFP-growth [14] وخوارزمية FP-growth [7] الأساسية كما هو موضح في الشكل 11 و12.

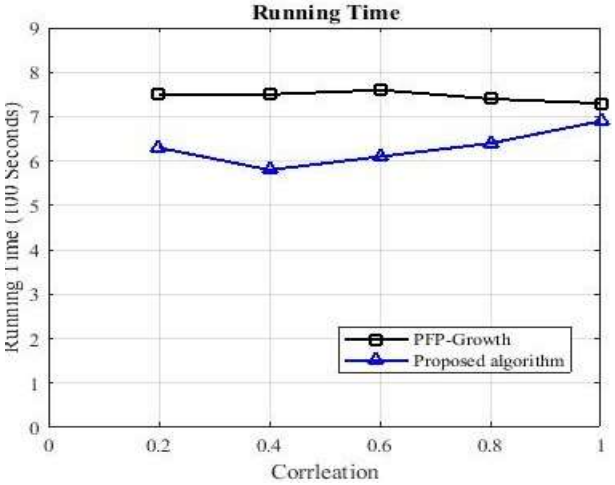
والشكل 13 يبين أن الخوارزمية المقترحة أيضاً تحقق زمن تنفيذ أسرع بين الخوارزميات الأخرى PFP-growth [14]. مع عدم قدرة خوارزمية FP-growth [7] الأساسية التعامل مع قاعدة المعطيات T40I10D100K وقاعدة المعطيات Kosarak.



الشكل 12: زمن التنفيذ من أجل عتبات دعم مختلفة **(T40I10D100K).**

من الأشكال السابقة نجد أن الخوارزمية المقترحة تستغرق زمن تنفيذ أقل بين الخوارزميات الأخرى المذكورة سابقاً، وخصوصاً عند التعامل مع البيانات الضخمة مثل Kosarak.

تم ضبط معامل الترابط عند تقسيم العناصر المتكررة بالقيم التالية (0.20, 0.40, 0.60, 0.80, 1) وذلك من أجل تقييم تأثير معامل الترابط على أداء الخوارزمية المقترحة. الشكل 15 يوضح أن الخوارزمية المقترحة أكثر حساسية لمعامل ترابط البيانات من خوارزمية PFP-growth [14]. يبين الشكل 14 أن أفضل قيمة لمعامل الترابط هي القيمة 40. فإذا كانت قيمة معامل الترابط منخفضة فهذا يؤدي تقسيم العناصر بصورة عشوائية ويصعب تقسيم العناصر المتكررة الى مجموعات مستقلة عند ضبط قيمة معامل الترابط بقيمة كبيرة.

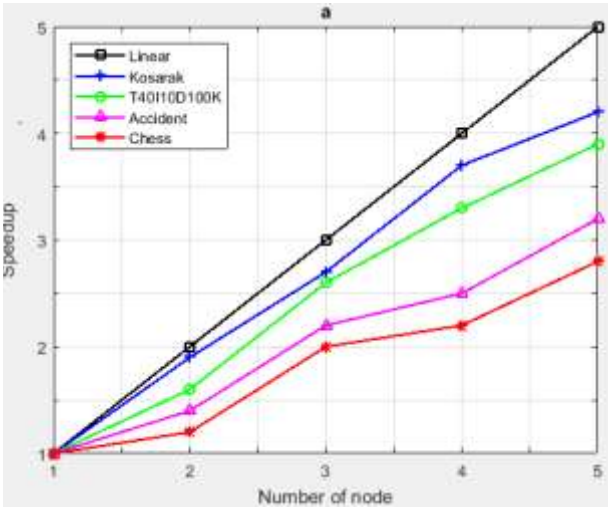


الشكل 14: تأثير معامل الترابط على الخوارزمية المقترحة.

ولتقييم أداء الخوارزمية التفرعية المقترحة سوف يتم استخدام عدة مقاييس، أهمها مقياس التسارع Speedup [6] [9] الموضح في المعادلة4، ويستخدم من أجل تقييم سرعة الخوارزمية التفرعية مقارنتاً بالخوارزمية التسلسلية المطبقة على عقدة واحدة.

$$\text{Speedup=S(n)} = \frac{T1}{Tn} \qquad (4)$$

حيث إن T_1: تمثل زمن التنفيذ التسلسلي للخوارزمية من خلال عقدة واحدة، أما T_n: تمثل زمن التنفيذ التفرعي للخوارزمية للتنقيب عن العناصر المتكررة وباستخدام نفس قاعدة المعطيات من خلال n عقدة حسابية. تم التنقيب عن العناصر المتكررة في قواعد المعطيات السابقة، باستخدام الخوارزمية المقترحة وانطلاقاً من عقدة حسابية واحدة حتى خمس عقد. من الشكل 15 نجد أن تسارع الخوارزمية المقترحة يزداد بشكل شبه خطي مع زيادة عدد العقد، وخصوصاً عند زيادة حجم قاعدة البيانات. وصلت قيمة التسارع من أجل قاعدة البيانات Kosarak الى 4,250 أي 85% (4,250/5=0.85) من قيمة التسارع المثالي. بشكل عام من الصعب الوصول إلى التسارع المثالي نتيجة كلفة الاتصال بين العقد الحسابية[9] .



الشكل 15: مقياس Speedup للخوارزمية المقترحة.

المقياس الأخر قابلية التوسع Scaleup [6] [9] وسوف يتم دراسة قابلية التوسع للخوارزمية التفرعية المقترحة من أجل توضيح ما يسمى بـ Workload Scaling factor من خلال قانون Gustafson's law الموضح في المعادلة5، وينصّ على أن زيادة حجم المسألة المدروسة يتناسب طرداً مع زيادة عدد العقد الحسابية مع بقاء الجزء التسلسلي من المسألة المدروسة ثابت عند تلك الزيادة. ونقول أن الخوارزمية تتمتع بقابلية التوسع ((Scalable algorithm عندما يتناسب زمن التنفيذ مع زيادة عدد العقد الحسابية المستخدمة وحجم المسألة المدروسة[13] [24] .

$$\text{Speedup=S(n)} = \frac{T1}{Tn} = 1 + (n-1) * p^* \qquad (5)$$

حيث إن T_1: تمثل زمن التنفيذ التسلسلي للخوارزمية من خلال عقدة واحدة، أما T_n: تمثل زمن التنفيذ التفرعي للخوارزمية على n عقدة حسابية، و n: عدد العقد الحسابية، أما P^*: تمثل نسبة الجزء التفرعي من المسألة المدروسة.

لتقييم قابلية التوسع للخوارزمية المقترحة عند التعامل البيانات الضخمة، تم زيادة أحجام قواعد المعطيات الأكبر حجماً من خلال تكرار البيانات (فتم زيادة حجم قاعدة المعطيات T40I10D100K من 30MB حتى 120MB، ومن 40MB حتى 160MB لقاعدة المعطيات Kosarak)، وبالمقابل تم زيادة عدد العقد من 1 إلى 2 ومن ثم 4 عقد، ولم يتم استخدام قاعدة المعطيات Chess و Accident وذلك لصغر حجمها. والجدول 8 والجدول 9 يبين زمن التنفيذ وقيم مقياس التسارع وفق قانون Gustafson's law.

الجدول8: زمن التنفيذ وقيم التسارع Speedup بالاعتماد على قانون Gustafson من أجل قاعدة البيانات (T40I10D100K).

Problem size (MB)	Execution Time(second)				Speedup	
T40I10D100K	T_1	T_2	T_4		S(2)	S(4)
30	466	241	118		1.93	3.94
60	1020	511	257		1.99	3.96
120	2134	1072	535		1.99	3.98

من الجدول 8 نجد أنه عندما تم معالجة قاعدة البيانات T40I10D100K وبحجم 60MB ومن خلال عقدتين حسابيتين فإن زمن التنفيذ 511sec وعند زيادة الحجم الى 120MB وزيادة العقد الحسابية الى الى أربع عقد فإن زمن التنفيذ قريب من 535sec، وبالتالي فإن قيم التسارع تتراوح بين 1.93 حتى 3.98 عند زيادة حجم قاعدة المعطيات من 30MB الى 120MB، مع زيادة عدد العقد الحسابية من 2 حتى 4.

الجدول 9: زمن التنفيذ وقيم التسارع Speedup بالاعتماد على قانون Gustafson من أجل قاعدة البيانات **(**Kosarak).

Problem size (MB)	Execution Time(second)				Speedup	
Kosarak	T_1	T_2	T_4		S(2)	S(4)
40	1178	599	297		1.96	3.96
80	2388	1208	600		1.97	3.98
160	4860	2440	1217		1.99	3.99

ونفس النتائج تنطبق على الجدول 9 فعند معالجة قاعدة البيانات Kosarak وبحجم 80MB ومن خلال عقدتين حسابيتين فإن زمن التنفيذ 1208sec وعند زيادة الحجم الى 160MB وزيادة العقد الحسابية الى الى أربع عقد فإن زمن التنفيذ قريب من 2440sec، وبالتالي فإن قيم التسارع تتراوح بين 1.96 حتى 3.99 عند زيادة حجم قاعدة المعطيات من 40MB الى 160MB، مع زيادة عدد العقد الحسابية من 2 حتى 4. وواضح من الجدولين السابقين أن الخوارزمية المقترحة قادرة على التكيف مع زيادة أحجام قواعد المعطيات بالاعتماد على زيادة عدد العقد الحسابية في منصة هادوب والبنية MapReduce.

المقياس الأخير هو كلفة (cost) الخوارزمية التفرعية والتي تساوي جداء زمن التنفيذ بعدد العقد الحسابية المستخدمة [13] [24]، ويستخدم هذا المقياس في برهان كفاءة عملية التفريع عند تحقق المعادلة6، أي يجب انجاز n عملية على n عقدة حسابية على الأقل.

$$\text{CP(n)} = n * T_n \geq T_1 \qquad (6)$$

من الجدول 10 نجد أن كلفة الخوارزمية التفرعية المقترحة تحقق المعادلة6 في كل الحالات.

الجدول 10: مقياس الكلفة **Cost** للخوارزمية المقترحة.

Dataset	T_1(sec)	T_5(Sec)	CP(5)=5*T_5
Chess	162	35	175
Accident	197	42	210
T40I10D100K	466	95	475
Kosarak	1178	240	1200

9. الاستنتاجات والتوصيات:

تم في هذا البحث دراسة لخوارزميات التنقيب عن العناصر المتكررة خاصة التطويرات الموزعة لخوارزمية FP-growth، حيث تم دراسة إيجابيات وسلبيات كل منها. ومن ثم تم اقتراح خوارزمية للتنقيب عن العناصر المتكررة بالاعتماد على المخطط الموجه وتم تطبيق الخوارزمية المحسنة على بنية هادوب الموزعة بهدف تمكين الخوارزمية المقترحة من التعامل مع البيانات الضخمة، ويمكن تلخيص النتائج كمايلي:

1. تخفيض عدد العقد والمساحة المطلوبة لتخزين مداولات قاعدة البيانات بمقدار 50% وذلك من خلال استبدال بنية الشجرة بالمخطط الموجه.
2. تسريع عملية التنقيب عن العناصر المتكررة بنسبة تتجاوز 60% في بعض الأحيان وذلك من خلال التخلص من بناء شجرة FP-tree الشرطية، والاعتماد على المخطط الموجه.
3. تم تخفيض التعقيد الحسابي والاتصال بين عقد التجمع وارسال المداولات بين العقد الحسابية بشكل متكرر من خلال توزيع المهام الحسابية بشكل متوازن بين العقد الحسابية وذلك باستخدام معامل الترابط بين العناصر المتكررة وتجميعها ضمن قائمة واحدة بهدف تقسيم المداولات بين العقد الحسابية.
4. أكدت النتائج التجريبية والتي طُبقت على قواعد معطيات مختلفة سواء كانت متفرقة أو كثيفة، ذات أحجام كبيرة أو صغيرة تحسن في أداء الخوارزمية المقترحة خاصة عند التعامل مع قواعد البيانات ذات الأحجام الكبيرة والتي عجزت العديد من الخوارزميات السابقة التعامل معها.
5. تتمتع الخوارزمية المقترحة بتسارع شبه خطي في معظم الحالات المدروسة سابقاً، كما أنها تتمتع بقابلية التوسع.

التوصيات:

على الرغم من فعالية الخوارزمية المقترحة، الا أنه لازال هناك العديد من التحسينات المستقبلية أهمها:

1. تمكين الخوارزمية المقترحة من التعامل مع تيار بيانات مستمرة (Stream Data) وذلك من خلال تقليص عدد مرات مسح قاعدة المعطيات الى مسح واحد.
2. تسريع عملية تجميع المداولات المتشابهة وذلك من خلال استخدام تقنيات LSH-based Partitioning.

المصادر:

[1] Agrawal, R., & Srikant, R. 1994. **"Fast algorithm for mining association rules in large databases"**. In Proceedings of 20th VLDB conference (pp. 487–499).

[2] Apache Hadoop. http://hadoo p. apach e.org/. Accessed 8 Oct 2019.

[3] Apiletti D., Baralis E., Cerquitelli T., Garza P., Pulvirenti F., and Venturini L., 2017, **"Frequent itemsets mining for big data: a comparative analysis,"** Big Data Research, vol. 9, pp. 67–83.

[4] Alhamodi A. A. G. and Lu S., 2016**," MRFP: Discovery Frequent Patterns Using MapReduce Frequent Pattern Growth**", IEEE, International Conference on Network and Information Systems for Computers, page 298-301, 978-1-4673-8838-2/16 $31.00©2016 IEEE, DOI 10.1109/ICNISC.2016.37.

[5] Dahdouh K., Dakkak A., Oughdir L. and Messaoudi F., 2019. '**Large-scale e-learning recommender system based on Spark and Hadoop'.** Springer, Journal of big data (2019) 6:2, http://doi.org/10.1186/s40537-019-0169-4.

[6] Gassama A.D.D., Camara F. and Ndiaye S.2017**," S-FPG: A Parallel Version of FP-growth Algorithm under Apache Spark ™"**, the second IEEE International Conference on Cloud Computing and Big Data Analysis, 978-1-5090-4499-3/17/$31.00 ©2017 IEEE.

[7] Han, J., Pei, J. and Yin .2000 **'Mining frequent patterns without candidate generation'**, in Proceedings of the ACM-SIGMOD Conference Management of Data, Vol. 29, No. 2, pp.1–12.

[8] Han, J., Pei, J., Yin, Y. and Mao, R. 2004 **'Mining frequent patterns without candidate generation: a frequent-pattern tree approach'**, Data Mining and Knowledge Discovery, Vol. 8, No. 1, pp.53–87.

[9] Ji S., Zhang D. and Zhang L, 2016**," Paths sharing based FP-growth data mining algorithms**", IEEE, 978-1-5090-2860-3/16/$31.00 ©2016 IEEE.

[10] Jiang H. and Meng H., 2017**," A Parallel FP-growth Algorithm Based on GPU"**, the Fourteenth IEEE International Conference on e-Business Engineering, 978-1-5386-1412-9/17 $31.00©2017IEEE, DOI 10.1109/ICEBE. 2017.24.

[11] Jian L., Wang C., Liu Y., Liang S., Yi W., and Shi Y., 2013, **"Parallel data mining techniques on graphics processing unit with compute unified device architecture (cuda),"** The Journal of Supercomputing, vol. 64, no. 3, pp. 942–967.

[12] Kulkarni P. G. and Khonde S. R., 2017, **"HDFS Framework for Efficient Frequent Itemset Mining Using MapReduce"**, IEEE, 978-1-5090-4264-7/17/$31.00 ©2017 IEEE.

[13] Kwiatkowski J. and Olech L. P., 2016**, "Scalability Model Based on the Concept of Granularity",** arXiv:1604.00554v1[cs. DC], 2 Apr 2016.

mining association rules". IEEE Transactions on Knowledge and Data Engineering, 9(5), 813–825.

[20] Shirke D., Varshney D. 2016**," A Survey on Techniques of Parallel Mining of Frequent Itemsets Using MapReduce"**, International Journal of Advance Research in Computer Science and Management Studies, ISSN: 2321 -7782 (Online).

[21] Tanbeer S.K., Chowdhury F.A., Jeong B. and Lee, Y-K. 2008 **"Efficient single-pass frequent pattern mining using a prefix-tree"**, in Information Sciences, Vol. 179, No. 5, pp.559–583.

[22] Tehreem A., Khawaja S.G., Akram M.U., Khan S. A. and Ali M., 2017**," Parallel Architecture for Implementation of Frequent Itemset Mining Using FP-growth"**, IEEE, 2017 International Conference on Signals and Systems (ICSigSys), 978-1-5090-6748-0/17/$31.00 ©2017 IEEE.

[23] Xun Y., Zhang J., Qin X. and Zhao X.,2016,**"FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters",** IEEE**,** 1045-9219 (c) 2016 IEEE.

[24] Woodside M. and Jogalekar P., 2000,**" Evaluating the Scalability of Distributed Systems"**, IEEE Transactions on Parallel and Distributed Systems, vol. 11, NO. 6, June2000, pp.589-603.

[25] FIMI 2004, FIMI Repository [online] http://fimi.cs.helsinki.fi/data Accessed 8 Oct 2019.

[14] Li H., Wang Y., Zhang D., Zhang M., and Chang E. Y., 2008**," PFP: Parallel FP-Growth for Query Recommendation"**. In Proceedings of the 2008 ACM conference on Recommender systems.

[15] Makanju A., Farzanyar Z., An A., Cercone N., Hu Z. Z. and Hu Y., 2016**," Deep Parallelization of Parallel FP-growth Using Parent-Child MapReduce", IEEE,** International Conference on Big Data, pp. 1422- 1431.

[16] Ma Z., Yang J., Zhang T. and Liu F. 2016**," An Improved Eclat Algorithm for Mining Association Rules Based on Increased Search Strategy"**, International Journal of Database Theory and Application, pp.251-266, Vol.9, ISSN: 2005-4270 IJDTA, No.5.

[17] Martire I., Da Silva P. N., Plastino A. and Fabris F., 2017**,"A novel probabilistic Jaccard distance measure for classification of sparse and uncertain data"**, Symposium on Knowledge Discovery, Mining and Learning, KDMILE 2017.

[18] Nikam P. V., Deshpande D. S., 2018**," New approach in Big Data Mining for frequent itemset using mapreduce in HDFS"**, IEEE, 3rd International Conference for Convergence in Technology (I2CT), 978-1-5386-4273-3 /18/ $31.00 ©2018 IEEE.

[19] Park, J. S., Chen, M. S., & Yu, P. S. 1997. **"Using a hash-based method with transaction trimming for**

Frequent Itemset Mining Based on Development of FP-growth Algorithm and Use MapReduce Technique

Zakria Mahrousa [1]**,** ***Dima Mufti Alchawafa*** [2]**, and** ***Hasan Kazzaz***[3]*****

[1] Department of Computer Engineering, Faculty of Electrical and Electronic Engineering, University of Aleppo, Syria, Rmahrousa@hotmail.com.

[2] Department of Computer Engineering, Faculty of Electrical and Electronic Engineering, University of Aleppo, Syria, dima.mufti@gmail.com.

3 Department of Computer Engineering, Faculty of Electrical and Electronic Engineering, University of Aleppo, Syria, kazzazhasan@gmail.com.

* Corresponding author: Hasan Kazzaz, kazzazhasan@gmail.com.

Published online: 31 March 2021

Abstract— The Finding of frequent itemset in big data is an important task in data mining and knowledge discovery. The exponential daily growth of data, called "Big Data", mining frequent patterns from the huge volumes of data has many challenges due to memory requirement, multiple data dimensions, heterogeneity of data and so on. The complexities related to mining frequent item-sets from a Big Data can be minimized by using Modified FP-growth algorithm and parallelizing the mining task with Map Reduce framework in Hadoop. In this paper, a modified FP-growth based on directed graph with Hadoop framework will reduce the execution time for the massive database and works efficiently on number of nodes (computers). The algorithm was tested, our experimental results demonstrated that the proposed algorithm could scale well and efficiently process large datasets. In addition, it achieves improvement in memory consumption to store frequent patterns and time complexity.

Keywords— Association Rule Mining (ARM), Big Data , Map-Reduce, FP-growth, Directed Graph.