Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Optimización de reglas de asociación generalizadas y jerárquico-temporales

Tesis de Licenciatura en Ciencias de la Computación

Emiliano Galimberti, Ignacio Manuel Fernández

Directora: Cecila Ana Ruz

Buenos Aires, 2022

Optimización de reglas de asociación generalizadas y jerárquico-temporales

Reglas de asociación es una técnica de minería de datos no supervisada que permite generar hipótesis a partir de los datos. Esta técnica consiste en encontrar todos los subconjuntos de items que ocurren frecuentemente en una base de datos, para luego generar reglas que indiquen la influencia y relación entre los elementos de los subconjuntos conseguidos anteriormente. El dominio de problema más común son las compras de supermercado.

Las reglas de asociación generalizadas son una extensión de las reglas de asociación. Éstas aprovechan el uso de una taxonomía predefinida sobre los datos para obtener información más abstracta y compacta. La taxonomía es definida por un experto del dominio o por un proceso automático. En el caso de una base de datos transaccional como las compras del supermercado, la taxonomía puede ser la categorización de los productos.

Otra extensión son las reglas de asociación temporales. Éstas buscan relacionar el espacio temporal con los items de la base de datos y utilizar esta información para obtener los subconjuntos que ocurren frecuentemente en distintos períodos de tiempo, sin descartarlos en caso de no cumplir con la frecuencia deseada en la totalidad de la base de datos. Vamos a utilizar una jerarquía temporal que nos ayudará a buscar reglas por períodos granulares.

En el presente trabajo se investigaron las soluciones existentes para este problema y las extensiones mencionadas. Luego, se llevaron a cabo implementaciones de cada una de ellas, agregando técnicas de optimización apoyadas por experimentación para luego concluir con la creación de un algoritmo que combina las reglas generalizadas con las temporales.

Finalmente, estos algoritmos se disponibilizarán mediante una librería pública en Python para su uso académico y profesional.

Palabras claves: Reglas de asociación - Reglas de asociación Generalizadas - Reglas de asociación Temporales.

Agradecimientos

A Cecilia, por el tiempo dedicado y su predisposición para ayudarnos a lo largo de todo el trabajo. Su pasión por la materia fue el combustible de esta tesis.

A nuestras familias, por apoyarnos a lo largo de todo este viaje. Por sus enseñanzas de no rendirse, y alcanzar nuestras metas.

A nuestros amigos, por multiplicar las felicidades y dividir las tristezas. En especial a *Lionhood* y a los *Finwinitos* que supieron apoyarnos ante este desafío final.

A Nacho. A Emi.

Índice

1. Introducción

La minería de datos es el proceso de extracción de patrones o información interesante de grandes bases de datos. Existen múltiples funcionalidades que se pueden aplicar a una base de datos como la clasificación, predicción, análisis de tendencias, etc.. Las técnicas de minería de datos se pueden separar en dos grupos, las supervisadas y no supervisadas. Las primeras parten de un conjunto de datos previamente etiquetados, que toman los algoritmos para poder predecir nuevos datos. En cambio en el segundo grupo de técnicas mencionadas, los algoritmos toman los datos sin etiquetas previas y su objetivo es contribuir a la comprensión de los datos bajo análisis. En esta tesis nos vamos a centrar en una técnica no supervisada llamada reglas de asociación [Agrawal et al., 1993].

Esta técnica viene siendo estudiada desde hace varios años y el dominio de problema más común son las compras de supermercado, conocido como *Market Basket Analysis* pero también tiene otras aplicaciones en el área de medicina [Şerban et al., 2006] y biología [Gupta et al., 2006].

Las reglas de asociación aplicadas a las compras del supermercado permiten extraer conclusiones respecto de cómo se relacionan ciertos productos que son adquiridos en forma conjunta, es decir, nos indican el grado de atracción de los productos que se compran en una misma transacción. A partir de esto se pueden adquirir patrones para generar promociones o llevar a una reorganización de la exposición de los productos.

Tid	Items
1	Pan, Leche
2	Pan, Pañales, Cerveza, Huevos
3	Leche, Pañales, Cerveza, Gaseosa
4	Pan, Leche, Pañales, Cerveza
5	Pan, Leche, Pañales, Gaseosa

Ejemplos de Reglas de Asociación [a]
{ Pañal } ⇒ { Cerveza }
{ Lecha, Pan } ⇒ { Huevos, Gaseosa }
{ Cerveza, Pan } ⇒ { Leche }

[a] La implicación es coocurrencia no causalidad

Cuadro 1: Ejemplo de transacciones de supermercado y reglas generadas

Definición 1. *Sea* $\mathcal{I} = \{i_1, i_2, ..., i_m\}$ *un conjunto de literales llamados items.*

En el ejemplo de las transacciones del supermercado presentadas en el Cuadro 1, $\mathcal{I}$ = { Pan, Leche, Pañales, Cerveza, Huevos, Gaseosa }

Definición 2. *Sea* $\mathcal{D}$ *el conjunto de transacciones donde cada transacción* T *es un conjunto de items tal que* $T \subseteq \mathcal{I}$.

Volviendo al Cuadro 1, una transacción T puede ser la del Tid 2 = { Pan, Pañales, Cerveza, Huevos }

Definición 3. *Llamamos* ***itemset*** *a un conjunto de items* X *tal que* $X \subseteq \mathcal{I}$.

Definición 4. *El* ***support count*** *(*σ*) de un itemset* X *es una función* $\sigma : 2^{\mathcal{I}} \rightarrow \mathbb{N} \cup \{0\}$ *que representa la cantidad de transacciones en* $\mathcal{D}$ *que contienen a* X.

Ejemplo.

$$\sigma(\{Pan, Leche\}) = 3$$

Ya que $\{Pan, Leche\}$ aparecen juntos en las transacciones con Tid $= \{1, 4, 5\}$.

Definición 5. *El **soporte** de un itemset X es la fracción de transacciones que contienen a X es decir,*

$$Soporte(X) = \frac{\sigma(X)}{|\mathcal{D}|}$$

Ejemplo.

$$soporte(\{Pan, Leche\}) = \frac{3}{5}$$

Ya que $\{Pan, Leche\}$ aparecen juntos en las transacciones con Tid $= \{1, 4, 5\}$ de un total de 5 transacciones.

Definición 6. *Llamamos **itemset frecuente** a los itemsets que tienen soporte mayor a un umbral establecido por el usuario.*

Definición 7. *Una regla de asociación es una implicación de la forma $X \Rightarrow Y$ donde $X \subseteq \mathcal{I}$, $Y \subseteq \mathcal{I}$ y $X \cap Y = \emptyset$.*

Definición 8. *Decimos que la regla $X \Rightarrow Y$ tiene **soporte** $s\,\%$ en el conjunto de transacciones $\mathcal{D}$ si $s\,\%$ de las transacciones en $\mathcal{D}$ contienen a $X \cup Y$. Es decir,*

$$soporte(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{|\mathcal{D}|}$$

Notar que el soporte de la regla es igual al soporte del itemset que la compone.

Ejemplo.

$$soporte(\{Pan \Rightarrow Leche\}) = soporte(\{Pan, Leche\}) = \frac{3}{5}$$

Esta medida nos indica cuándo una regla puede interesarnos y cuándo no, en base a la frecuencia de aparición en los tickets de compras. Cuando el soporte es bajo se pueden rechazar conjuntos que no son de interés y refinar la búsqueda de reglas.

Ejemplo.

$$\textbf{soporte}(\{Cerveza, Pan\} \Rightarrow \{Leche\}) = \frac{1}{5} = 0.2$$

Definición 9. *Decimos que la regla $X \Rightarrow Y$ tiene **confianza** $c\,\%$ si $c\,\%$ de las transacciones en $\mathcal{D}$ que contienen a X también contienen a Y. Es decir,*

$$\textbf{confianza}(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

La confianza es un estimador de la probabilidad de que ocurra el lado derecho de la regla considerando el total de las transacciones que contienen al lado izquierdo, es decir, estima la probabilidad condicional $P(Y|X)$.

Ejemplo.

$$confianza(\{Cerveza, Pan\} \Rightarrow \{Leche\}) = \frac{1}{2} = 0.5$$

Definición 10. *Decimos que una regla de asociación es válida y pertenece al conjunto de reglas solución si posee un soporte y una confianza mayor o igual a los umbrales establecidos por el usuario.*

En el último tiempo se presentaron trabajos que extendían el dominio del problema de las reglas de asociación. No sólo ampliando el origen de los datos y la naturaleza del negocio sino también incorporando nuevos parámetros a tener en cuenta al momento de generar la asociación pertinente. En este trabajo veremos dos extensiones que involucran el tiempo y la taxonomía de los elementos por separado y luego su combinación.

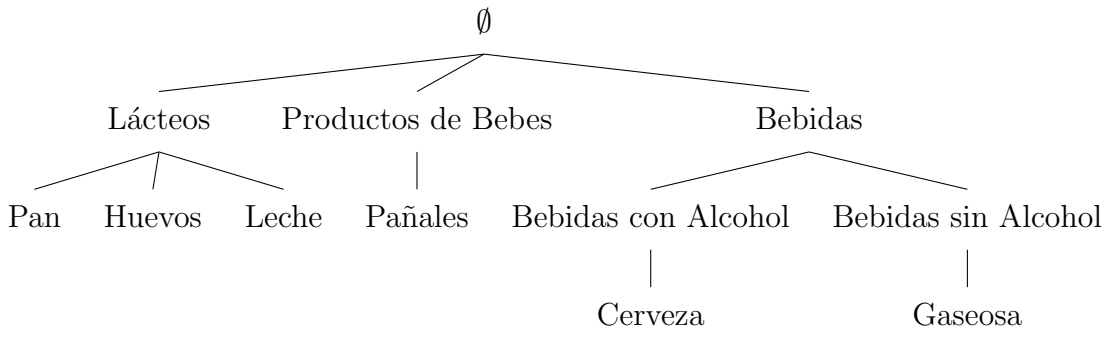
1.1. Reglas de asociación generalizadas

Las reglas de asociación generalizadas [Srikant and Agrawal, 1995] tienen por objetivo encontrar reglas de asociación en un conjunto de transacciones $\mathcal{D}$ con la información adicional de una o más taxonomías de los ítems en $\mathcal{I}$.

Definición 11. *Llamamos $\mathcal{G}$ a un grafo dirigido sin ciclos que representa un conjunto de taxonomías.*

Un eje en $\mathcal{G}$ representa un relación $es - padre$. Si existe un eje en $\mathcal{G}$ de p a c decimos que p es padre de c y c un hijo de p (p representa una generalización de c).

Ejemplo. Volviendo al ejemplo del supermercado, los productos $\mathcal{I}$ = { Pan, Leche, Pañales, Cerveza, Huevos, Gaseosa } son las hojas de la taxonomía y se puede extender $\mathcal{I}$ con los siguientes productos generalizados: { Lácteos, Productos para Bebes, Bebidas con Alcohol, Bebidas sin Alcohol }. La taxonomía resultante figura en el cuadro 2.



Cuadro 2: Ejemplo de una taxonomía de productos de un supermercado

En una base de datos real de un supermercado las hojas por lo general son los productos con marca y su generalización es la categoría a la que pertenece en el catálogo.

Definición 12. *Llamamos $\hat{x}$ como un ancestro de x (y x un descendiente de $\hat{x}$) si existe un eje de $\hat{x}$ a x en la clausura transitiva del grafo $\mathcal{G}$. En otras palabras existe un camino de $\hat{x}$ a x en $\mathcal{G}$. Como el grafo no tiene ciclos un ítem no puede ser ancestro de si mismo.*

Ejemplo. *Bebîdas* es ancestro de Bebidas con Alcohol, Bebidas sin Alcohol, Cerveza y Gaseosa.

Definición 13. *Una regla de asociación generalizada es una implicación de la forma $X \Rightarrow Y$, donde*

- $X \subset I$, $Y \subset I$
- $X \cap Y = \emptyset$
- *Ningún ítem en Y es un ancestro de algún ítem en X.*

Una regla se dice que es *generalizada* cuando X e Y contienen ítems de cualquier nivel de la taxonomía. Sin embargo existe una restricción para Y. Ningún ítem en Y puede ser un ancestro de algún ítem en X ya que la regla $x \Rightarrow ancestro(x)$ es trivialmente cierta con 100 % de confianza y por ende redundante.

Descubrir todas las reglas generalizadas produce un conjunto de reglas muy grandes y muchas veces no alcanza con ajustar el umbral de soporte y confianza mínimos. Este tipo de reglas requieren refinamientos ya que muchas de ellas son redundantes y trivialmente ciertas.

Para filtrar estas reglas no deseadas, basándonos en la información de la taxonomía utilizaremos una medida de interés llamada R-interesante. Veamos un ejemplo para explicar el concepto:

Sea la regla *Leche* $\Rightarrow$ *Cereal* (8 % de soporte, 70 % confianza).

- *Leche* es *padre de Leche Descremada*
- 25 % de las ventas de *Leche* son de *Leche Descremada*

- Si se mantiene la proporción mencionada en el punto anterior, lo esperable es *Leche Descremada* $\Rightarrow$ *Cereal* tenga 2 % soporte y 70 % de confianza ya que cada vez que aparece Leche Descremada en una transacción también aparece la Leche.

Si el soporte y la confianza real de la regla *Leche Descremada* $\Rightarrow$ *Cereal* es 2 % y 70 % respectivamente, la regla puede ser considerada redundante porque no transmite información adicional y es menos general que la primera regla mencionada. Capturamos esta noción de *interés* diciendo que solo queremos hallar las reglas cuyo suporte sea mayor o igual a R veces que el soporte esperado o cuya confianza sea mayor o igual a R veces que la confianza esperada donde R es un valor especificado por el usuario. En el caso del ejemplo anterior la regla *Leche Descremada* $\Rightarrow$ *Cereal* sería 2-interesante si su soporte fuera 4 %.

1.2. Reglas de asociación temporales

Para introducir el nuevo problema a resolver, vamos a tomar como ejemplo un reducido registro de compras de un año en el Cuadro 3. Cada compra está compuesta por su número de transacción (TID), fecha e itemset correspondiente (elementos comprados) .

TID	Fecha	Itemset
1	20 / 4	C, E, G, K
2	21 / 4	D, F, G
3	27 / 4	C, T
4	3 / 6	D, F, G, L
5	16 / 6	A, C, D, F
6	21 / 9	A, C, J, K
7	24 / 9	A
8	12 / 12	A, B

Cuadro 3: Base con timestamp

Regla	Soporte	Confianza
$F \Rightarrow D$	37 %	100 %
$D \Rightarrow F$	37 %	100 %

Cuadro 4: Reglas obtenidas con Apriori

Tomando un soporte mínimo de 35 % y una confianza mínima de 60 % (valores usados sólo para obtener ejemplos que sirvan a la demostración) podemos obtener únicamente dos reglas que se muestran en el Cuadro 4. Estas reglas son válidas teniendo en cuenta toda la base, lo que correspondería temporalmente a todo el año. Sin embargo, si tuviésemos en cuenta sólamente el último trimestre, (Octubre, Noviembre y Diciembre) podemos ver dos nuevas reglas ($A \Rightarrow B$ y $B \Rightarrow A$) ambas con soporte y confianza 100 %. Lo mismo ocurre si tomamos sólo el segundo trimestre, podríamos encontrar reglas como ($G \Rightarrow D$) o ($D, F \Rightarrow G$), ambas con soporte 40 % y confianza 60 %. Ya si nos quedamos solamente con un mes, como por ejemplo Junio, la cantidad de reglas aumenta significativamente junto con su soporte y confianza. Es de esperar que esto ocurra en la medida que acortemos el período de tiempo en consideración.

Observemos las reglas obtenidas previamente. Si bien estas reglas son válidas si tenemos en cuenta toda la base (o sea, todo el año), es importante observar que esto no se cumple para el

primer, tercer o cuarto trimestre del año. Incluso tampoco se cumple para todos los meses que integran el segundo trimestre, sino sólo para el mes de Junio.

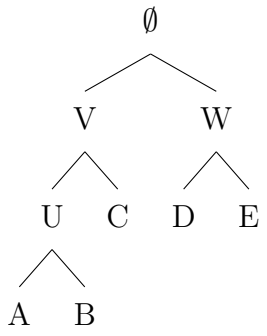
De esta manera podemos observar cómo las reglas obtienen una nueva interpretación al identificar y distinguir los períodos de tiempo donde poseen mas injerencia.

1.3. Reglas de asociación generalizadas jerárquico-temporales

Las reglas de asociación generalizadas jerárquico-temporales son un nuevo tipo de reglas de asociación que combinan dos tipos las reglas: Las generalizadas y las jerárquico-temporales. Para poder generar este nuevo tipo de reglas se tiene que agregar al conjunto de datos una taxonomía y un conjunto de timestamps asociados a cada transacción, obteniendo como resultado el Cuadro 7. Esta es una base de datos con timestamp (Cuadro 5) cuyas transacciones fueron expandidas con la información de la taxonomía (Cuadro 6).

TID	Fecha	Itemset
1	20 / 4	A, B
2	21 / 4	C, D, E
3	27 / 4	A, C, D
4	3 / 6	B, D

Cuadro 5: Base con timestamp



Cuadro 6: Taxonomía

TID	Fecha	Itemset
1	20 / 4	A, B, U, V
2	21 / 4	C, D, E, V, W
3	27 / 4	A, C, D, U, W
4	3 / 6	B, D, U, V, W

Cuadro 7: Base con timestamp expandida

Definición 14. *Una regla de asociación generalizada jerárquico-temporal es una implicación de la forma* $X \Rightarrow Y$*, donde*

- $X \subset I$, $Y \subset I$
- $X \cap Y = \emptyset$
- *Ningún ítem en* Y *es un ancestro de algún ítem en* X*.*
- $X \cup Y$ *es frecuente en un período temporal en el conjunto de transacciones con timestamp*

1.4. Objetivos de la tesis

Para cada uno de los problemas mencionados anteriormente, (Reglas de asociación, Reglas de asociación generalizadas y Reglas de asociación temporales) vamos a proponer e implementar algoritmos (*Apriori, Cumulate y HTAR*). Estas implementaciones no sólo tendrán mejoras en el código (como cambios de estructuras de datos) sino que se añadirán mejoras propuestas en la presentación de esta tesis basadas en la paralelización de tareas [Agrawal and Shafer, 1996].

Además se creará un nuevo algoritmo (*HTGAR*) que combina *Cumulate* y *HTAR*, ofreciendo un nuevo acercamiento multinivel al problema original, generando las reglas generalizadas jerárquico-temporales.

Finalmente, se realizarán experimentaciones sobre todos los algoritmos, y sus modificaciones para evidenciar las mejoras logradas.

La implementación y mejora de estos algoritmos y sus respectivas experimentaciones estarán divididas entre los dos alumnos que desarrollaron esta tesis. En detalle:

- ***Apriori.*** [Agrawal et al., 1993] Este algoritmo resuelve el problema de generar las reglas de asociación y se va a implementar una versión que mejora su performance (mejora de tiempo de ejecución) utilizando técnicas de almacenamiento de datos y paralelismo. El objetivo es estudiar una serie de optimizaciones que servirán a futuro para los otros algoritmos.

  Esta sección será realizada por ambos alumnos.

- ***Cumulate.*** [Srikant and Agrawal, 1995] Este algoritmo resuelve el problema de generar las reglas de asociación generalizadas. Se va a implementar una versión optimizada del mismo. El objetivo es aprovechar la similitud que tiene con el algoritmo *Apriori* y aplicar las mismas optimizaciones realizadas para comparar la performance con la version original de *Cumulate*.

  Esta sección será realizada por Ignacio.

- ***HTAR.*** [Hong et al., 2016] Este algoritmo es una versión de *HTAR: Hierarchical Temporal Association Rule* que busca resolver la generación de reglas temporales.

  Dado que para su uso es necesario una *jerárquia temporal* que delimite los intervalos de tiempo por transacción, llamaremos a estas reglas *jerárquico-temporales*. Definiremos una jerarquía detallada posteriormente y aplicaremos las mismas optimizaciones basadas en la paralelización de los demás algoritmos, vistas en [Srikant and Agrawal, 1995]. Se detallarán las diferencias con el algoritmo original y veremos dos métodos para calcular el *Soporte Relativo* de cada itemset en cada gránulo temporal.

  Finalmente se harán experimentaciones para comparar y visualizar la diferencia entre los algoritmos optimizados y sus distintas versiones.

  Esta sección estará a cargo de Emiliano.

- ***HTGAR.*** Este algoritmo genera reglas generalizadas jerárquico-temporales, un nuevo tipo de reglas que combina las implementaciones de *Cumulate* y *HTAR*. El objetivo es tomar experimentos realizados previamente con cada algoritmo individual y aplicarlos al algoritmo combinado para estudiar la performance.

  Esta sección será realizada por ambos alumnos.

Además de los fines académicos de este trabajo, tenemos como objetivo final publicar una librería disponible para ser utilizada en Python con todos los algoritmos anteriormente descriptos para el uso de la comunidad.

2. Algoritmo *Apriori*

Dado un conjunto de transacciones T, el objetivo del descubrimiento de reglas de asociación es encontrar todas las reglas R que cumplen:

- soporte(R) $\geq$ soporte establecido por el usuario.
- confianza(R) $\geq$ confianza establecida por el usuario.

Se puede pensar en una aproximación por fuerza bruta, que consistiría en listar todas las posibles reglas de asociación, calcular el soporte, la confianza y eliminar las que no satisfacen los umbrales predefinidos. Veamos primero cuantos itemsets candidatos hay para 5 items distintos:

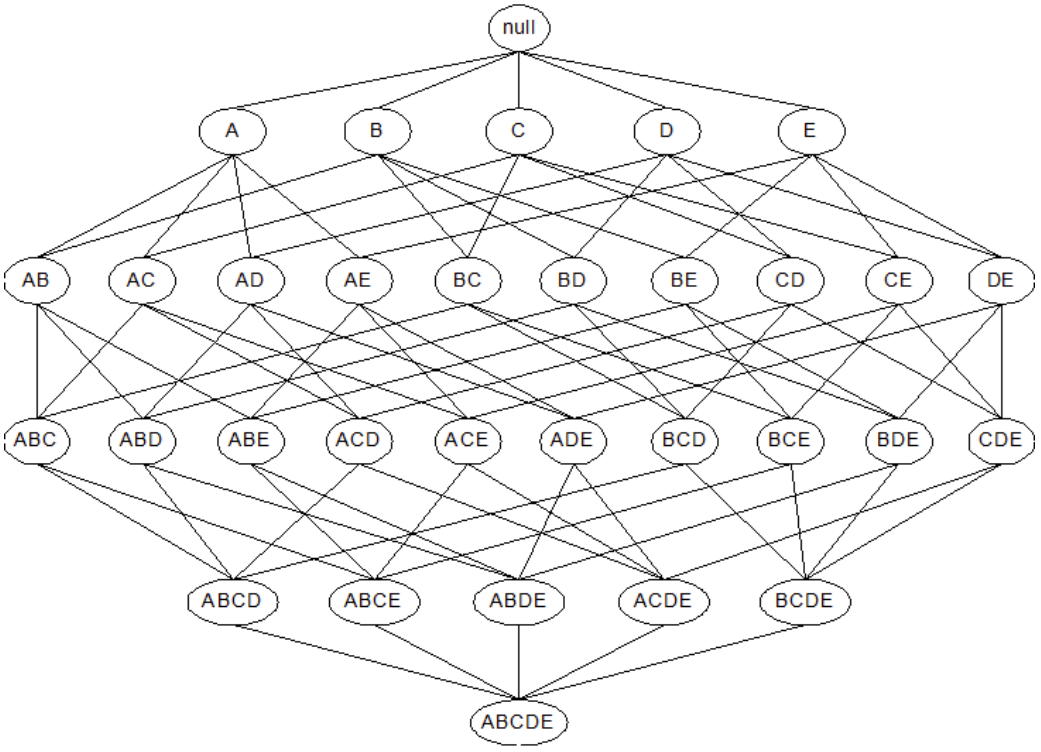


Figura 1: Itemset candidatos para 5 items distintos

Dados d items distintos del dataset, hay 2^d itemsets candidatos. Luego hay que contar el soporte de cada candidato barriendo las transacciones. Sea N el total de transacciones, w el largo máximo de una transacción y $M = 2^d$ candidatos, hacer este proceso de fuerza bruta tiene una complejidad algorítmica de $\mathcal{O}(NMw)$. El total de reglas R a generar es:

$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right] = 3^d - 2^{d+1} + 1$$

Esto es computacionalmente prohibitivo ya que la cantidad de reglas que se pueden generar es exponencial en el número de ítems. Una mejor aproximación a este problema es hacerlo en dos pasos: Primero generar los itemset frecuentes y luego generar las reglas. El algoritmo que vamos a estudiar a continuación tiene el nombre de *Apriori* [Agrawal and Srikant, 1994].

2.1. Características del algoritmo

El problema de descubrir todas las reglas de asociación puede descomponerse en dos subproblemas:

1. Encontrar todos los conjuntos de items que tengan soporte mayor o igual al especificado por el usuario.
2. Dado el conjunto de itemsets **frecuentes**, generar las reglas de asociación. Vamos a restringirnos a obtener las reglas con un solo consecuente. Esto permitirá reducir la cantidad de resultados.

La idea general del primer subproblema se puede ver en el siguiente pseudocódigo:

Algorithm 1 Apriori

Input *minsupp*, *Conjunto de transacciones* $\mathcal{D}$, *Conjunto de items* $\mathcal{I}$

```
1:  k ← 1
2:  L_1 ← {i ∈ I | soporte(i) ≥ minsupp}
3:  while L_k ≠ ∅ do
4:      C_k ← Generar candidatos de tamaño k a partir de L_k
5:      for t ∈ D do
6:          for c ∈ C_k do
7:              if c ⊆ t then
8:                  σ(c) ← σ(c) + 1
9:              end if
10:         end for
11:     end for
12:     L_k ← {c ∈ C_k | soporte(c) ≥ minsupp}
13:     k ← k + 1
14: end while
15: return ⋃_k L_k
```

Fuente:[Agrawal and Srikant, 1994]

2.2. Generación de candidatos

La generación de candidatos (C_k) es un paso vital en el descubrimiento de itemsets frecuentes (L_k). Para generarlos uno puede tomar L_{k-1} (ordenados lexicográficamente) y hacer la unión

consigo mismo. Al estar ordenados sólo queremos combinar los itemsets cuyos primeros $k-1$ items sean idénticos. Esto es para no generar el mismo candidato dos veces. Por ejemplo si queremos combinar $\{A, B, C\}$ con $\{A, C, D\}$ obtendríamos $\{A, B, C, D\}$. Si también combinamos $\{A, B, C\}$ con $\{A, B, D\}$ también obtendríamos $\{A, B, C, D\}$ generando un duplicado. Si los itemsets se ordenan y solo se genera una combinación cuando los primeros $k-1$ son idénticos se evitaría el problema de duplicar candidatos.

Si bien esta forma obtiene todas las combinaciones posibles de candidatos de tamaño k, dentro de ellos podría haber itemsets que no serán frecuentes porque no cumplen el siguiente principio:

Definición 15. ***Principio de Apriori.*** *Si un itemset es frecuente, entonces todos sus subitems son frecuentes.*

Este principio se mantiene gracias a la siguiente propiedad:

$$\forall X, Y : (X \subseteq Y) \Rightarrow soporte(X) \geq soporte(Y)$$

Luego para cada candidato $c \in C_k$ se puede revisar si existe un subconjunto c' de tamaño $(k-1)$ tal que $c' \notin L_{k-1}$, es decir que no sea frecuente. En caso de existir, c se quita de C_k. Este paso se llama *pruning* de candidatos. Veamos el pseudo-código:

Algorithm 2 Candidate-Generation

Input L_{k-1}

1: **for** $p \in L_{k-1}$ **do**
2: **for** $q \in L_{k-1}$ **do**
3: $C_k \leftarrow \{p.item_1, ..., p.item_{k-1}, q.item_{k-1} |\ p.item_i = q.item_i\ \forall i, 1 \leq i \leq k-2,\ p.item_{k-1} < q.item_{k-1}\}$ ▷ Al estar ordenados sólo queremos combinar los itemsets cuyos primeros $k-1$ items sean idénticos.
4: **end for**
5: **end for**
6: **for** $c \in C_k$ **do**
7: **for** $\{s \mid s \in subsets(c) \wedge |s| = k-1\}$ **do**
8: **if** $s \notin L_{k-1}$ **then**
9: remove c from C_k
10: **end if**
11: **end for**
12: **end for**
13: **return** C_k

Fuente:[Agrawal and Srikant, 1994]

2.3. Generación de reglas

Como mencionamos en la sección 2.2, el segundo subproblema de Apriori es generar todas las reglas a partir de los itemset frecuentes. La idea general es que si el itemset { A, B, C } es frecuente podemos determinar que la regla $\{A, B\} \Rightarrow \{C\}$ es válida si cumple que $conf = soporte(\{A, B, C\})/soporte(\{A, B\})$ es mayor o igual que la confianza mínima. Veamos en pseudocódigo como se obtienen las reglas:

Algorithm 3 Rule-Generation

Input $\bigcup_k L_k,\ minconf$

```
1:  R ← ∅
2:  for X ∈ ⋃_k L_k do
3:      for A ∈ subset(X) do
4:          B ← X − A
5:          if Confianza(A ⇒ B) ≥ minConf then
6:              R ← R ⋃ {(A ⇒ B)}
7:          end if
8:      end for
9:  end for
10: return R
```

2.4. Modificaciones Agregadas

2.4.1. Base de datos vertical

A pesar de las mejoras que realiza *Apriori* para descartar itemsets candidatos mediante la etapa de pruning, es muy costoso para el algoritmo leer todas las transacciones por cada iteración de la búsqueda de itemsets frecuentes de tamaño k. Una forma de calcular el soporte de un itemset de manera rápida sin tener que leer todas las transacciones es tener un diccionario como estructura de datos que permita indexar por los ítem y acceder a las transacciones en las que aparecen como un conjunto.

Para armar la estructura de datos se puede utilizar una técnica denominada *verticalizar* la base. La técnica consiste en leer todas las transacciones una sola vez y por cada una de ellas ir agregando al diccionario el ítem como clave y agregar al conjunto de valores el identificador de la transacción.

Tomando I el conjunto de items y D el conjunto de transacciones, donde cada transacción T es un conjunto de items, llamaremos $TID = \{1, 2, ..., n\}$ a la enumeración de las transacciones en T que servirán como identificadores. Notar que $|D| = |TID|$. El proceso de *verticalizar* la base de datos consiste en asignar a cada $i \in I$ un conjunto de identificadores de TID. Veámoslo en el siguiente ejemplo: Sea $I = \{A, B, C, D, E\}$ y $TID = \{1, 2, 3, 4\}$.

Tid	Itemset
1	A,C,D,E
2	A,B,C
3	B,C,D,E
4	A,C,D

Cuadro 8: Base Horizontal

Items	Tidsets
A	1,2,4
B	2,3
C	1,2,3,4
D	1,3,4
E	1,3

Cuadro 9: Base Vertical

Sea la función $t : I \rightarrow TID$ la que toma un ítem de I y devuelve un $TID' \subseteq TID$, las transacciones en las que aparece un $i \in I$ se calculan como $t(i) = TID'$. El soporte de un ítem individual i es $|t(i)|/|TID|$. Para calcular el soporte de un conjunto de items simplemente hay que calcular la intersección de las transacciones en las que aparecen y dividirlas por el total de transacciones.

Tomando el Cuadro 9, si se quiere calcular el soporte de $A \cup C$ puede hacerse de la siguiente forma: $soporte(A \cup C) = |t(A) \cap t(C)|/|TID| = |\{1,2,4\} \cap \{1,2,3,4\}|/4 = |\{1,2,4\}|/4 = 3/4$.

2.4.2. Paralelización del cálculo del soporte de los candidatos

Apriori es un algoritmo serial y por lo general las bases de datos que se utilizan para descubrir patrones con estas técnicas son muy grandes. Una ventaja que se puede aprovechar para mejorar la performance es usar la paralelización en la etapa del cálculo del soporte de candidatos. Después de generar los candidatos de tamaño k, el algoritmo calcula el soporte de cada uno de ellos y hasta que no termine no se pueden calcular los candidatos de tamaño $k+1$.

Sea $c \in C_k$, el proceso de calcular el soporte de c es independiente de cualquier otro proceso que calcula el soporte de un $c' \in C_k$ $t.q.$ $c \neq c'$. Llamemos P^i al proceso encargado de calcular el soporte de $c_{i_k} \in C_k$. La paralelización del algoritmo funciona de la siguiente manera para cualquier iteración $k > 1$:

1. Cada proceso P^i calcula el soporte del candidato $c_{i_k} \in C_k$ devolviendo el par $(c_{i_k},\ soporte(c_{i_k}))$
2. Cada resultado del proceso P^i se almacena en una lista $R = [(c_{i_k},\ soporte(c_{i_k}))|\ 1 < i < |C_k|]$
3. Cuando finalizan todos los procesos P^i se filtra de la lista R todos los pares que no cumplen el soporte mínimo. Luego de filtrar la lista R se proyecta la primer coordenada de los pares y se obtiene L_k

Veamos un ejemplo tomando el Cuadro 9 como base de datos. Con un soporte del 75 % los candidatos en $k = 2$ son $\{A,B\}, \{A,D\}, \{C,D\}$. Calcular el soporte de cada uno es un proceso independiente, entonces podemos paralelizar el cálculo del soporte de $\{A,B\}$, $\{A,D\}$ y $\{C,D\}$.

2.4.3. Paralelización de la generación de reglas

El problema de generar las reglas es menos costoso que descubrir los itemsets frecuentes ya que no requiere examinar los datos nuevamente. Sin embargo el soporte mínimo puede ser bajo resultando en un volumen grande de itemsets frecuentes para generar las reglas.

Dado un itemset frecuente l, la generación de reglas examina cada subconjunto no vacío a y genera una regla válida $a \Rightarrow (l-a)$ si y sólo si la confianza de la regla es mayor que la confianza mínima. La regla contiene soporte $soporte(l)$ y confianza $= soporte(l)/soporte(a)$, $|(l-a)| = 1$. Para generar las reglas en paralelo se debe distribuir a cada proceso P^i el trabajo de generar la regla dado un itemset $l \in \bigcup_k L_k$. La paralelización del algoritmo funciona de la siguiente manera

1. Cada proceso P^i genera un conjunto de reglas a partir de un itemset $l \in \bigcup_k L_k$ utilizando el algoritmo Rule-Generation 3
2. El resultado de cada proceso P^i se almacena es una lista R que contiene las reglas que superan la confianza mínima.
3. Cuando finalizan todos los procesos P^i el algoritmo devuelve R como resultado.

Veamos un ejemplo tomando el siguiente conjunto de itemset frecuentes $L_k = \{\{A, B\}, \{C, D\}\}$. Generar las reglas de asociación de cada itemset frecuente es un proceso independiente con lo cual podemos calcular la confianza de las reglas $\{A\} \Rightarrow \{B\}$ y $\{B\} \Rightarrow \{A\}$ (generadas a partir del itemset $\{A, B\}$) y en paralelo calcular la confianza de las reglas $\{C\} \Rightarrow \{D\}$ y $\{D\} \Rightarrow \{C\}$ (generadas a partir del itemset $\{C, D\}$).

3. Algoritmo *Cumulate*

En el paper [Srikant and Agrawal, 1995] se crean tres algoritmos que generan reglas de asociación generalizadas. Ellos son *Basic*, *Cumulate* y *EstMerge*. El concepto de *Basic* es simple: Escanear la base de datos y agregar a cada transacción los ancestros de cada ítem. Luego ejecutar *Apriori* y devolver las reglas generalizadas.

Cumulate es un algoritmo basado en *Basic* pero agrega un conjunto de optimizaciones que mejoran la performance del algoritmo. Nosotros vamos a implementar la versión original *Cumulate* y también una versión que le agrega otras mejoras para comparar resultados.

Antes de explicar el algoritmo vamos a formalizar el concepto de regla R-*interesante* en la siguiente sección.

3.1. Reglas R-interesantes

Definición 16. *Llamamos $\hat{Z}$ a un ancestro de Z si:*

- *Z, $\hat{Z}$ son conjuntos de items tal que $Z, \hat{Z} \subseteq I$*
- *Podemos obtener $\hat{Z}$ de Z reemplazando uno o más items en Z con sus ancestros.*
- $|Z| = |\hat{Z}|$

Las reglas $\hat{X} \Rightarrow Y$, $X \Rightarrow \hat{Y}$ o $\hat{X} \Rightarrow \hat{Y}$ son ancestros de la regla $X \Rightarrow Y$.

Definición 17. *Dado un conjunto de reglas llamamos $\hat{X} \Rightarrow \hat{Y}$ un ancestro* ***cercano*** *de $X \Rightarrow Y$ si:*

- *No existe regla $X' \Rightarrow Y'$ tal que $X' \Rightarrow Y'$ sea ancestro de $X \Rightarrow Y$ y*
- *$\hat{X} \Rightarrow \hat{Y}$ sea ancestro de $X' \Rightarrow Y'$*

(Definiciones similares se aplican a $\hat{X} \Rightarrow Y$ y $X \Rightarrow \hat{Y}$)

Consideremos la regla $X \Rightarrow Y$ y sea $Z = X \cup Y$. El soporte de Z es el mismo que el de la regla $X \Rightarrow Y$. Sea $Z = \{z_1, ...z_n\}$ y $\hat{Z} = \{\hat{z}_1, ..., \hat{z}_j, z_{j+1}, ...z_n\}$, $1 \leq j \leq n$ donde $\hat{z}_i$ es un ancestro de z_i. Definimos la siguiente formula como el soporte esperado de Z basado en $\hat{Z}$:

$$E_{\hat{Z}}[Pr(Z)] = \frac{Pr(z_1)}{Pr(\hat{z}_1)} \times ... \times \frac{Pr(z_j)}{Pr(\hat{z}_j)} \times Pr(\hat{Z})$$

Similarmente sea $E_{\hat{X} \Rightarrow \hat{Y}}[Pr(Y|X)]$ la confianza esperada de la regla $X \Rightarrow Y$. Sea $Y = \{y_1, ..., y_n\}$ y $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_j, y_{j+1}, ...y_n\}$, $1 \leq j \leq n$ donde $\hat{y}_i$ es un ancestro de y_i. Definimos la siguiente formula:

$$E_{\hat{X}\Rightarrow\hat{Y}}[Pr(Y|X)] = \frac{Pr(y_1)}{Pr(\hat{y}_1)} \times ... \times \frac{Pr(y_j)}{Pr(\hat{y}_j)} \times Pr(\hat{Y}|\hat{X})$$

Observación Sea $Pr(x)$ la probabilidad de que todos los items en X estén contenidos en una transacción. Luego el soporte de $X \Rightarrow Y = Pr(X \cup Y)$ y la confianza de $X \Rightarrow Y = Pr(X|Y)$

Definición 18. *Llamamos a la regla $X \Rightarrow Y$ **R-interesante** con respecto a su ancestro $\hat{X} \Rightarrow \hat{Y}$ si el soporte de la regla $X \Rightarrow Y$ es R veces mayor o igual que el soporte esperado basado en $\hat{X} \Rightarrow \hat{Y}$, o la confianza es R veces mayor o igual que la confianza esperada basada en $\hat{X} \Rightarrow \hat{Y}$*

Definición 19. *Dado un conjunto S de reglas y un mínimo nivel de interés R, la regla $X \Rightarrow Y$ es **interesante** en S si:*

- *No tiene ancestros o*
- *Es R-interesante con respecto a sus ancestros cercanos entre todos los ancestros interesantes.*

Definición 20. *Decimos que la regla $X \Rightarrow Y$ es **parcialmente interesante** en S si:*

- *No tiene ancestros o*
- *Es R-interesante con respecto a **al menos uno** de su ancestros cercanos entre todos los ancestros interesantes*

Los ancestros *interesantes* son todos los ancestros de un itemset cuyo soporte es R veces mayor o igual que el soporte esperado.

Formalización del problema. Dado un conjunto de transacciones D y un mínimo nivel de interés R especificado por el usuario, el problema de encontrar reglas de asociación con taxonomías es encontrar toda regla interesante que tenga soporte y confianza mayor o igual al especificado por el usuario. Para algunas aplicaciones se pueden hallar solamente las reglas parcialmente interesantes. Notar que si $R = 0$ se encuentran todas las reglas de asociación.

3.2. Características del algoritmo

Este algoritmo agrega varias optimizaciones al algoritmo *Basic* de [Srikant and Agrawal, 1995]. Consideremos el problema de decidir si una transacción T contiene a un itemset X. Esto requiere verificar para cada ítem $x \in X$ si x o un descendiente de él está presente en la transacción. La tarea se simplificaría si primero agregamos todos los ancestros de cada ítem de T en T. Llamemos T' a la transacción *expandida* de T. Ahora X tiene soporte en T si y solo si T' es un superset de X (Es decir, un set que contenga todos los elementos de X). Luego, una forma directa de encontrar todas las reglas de asociación generalizadas sería ejecutar el algoritmo Apriori de [Agrawal and Srikant, 1994] en las transacciones expandidas. Sin embargo el paper demuestra que se pueden agregar ciertas optimizaciones al algoritmo *Basic*. El conjunto de todas ellas forman el algoritmo *Cumulate*.

Cumulate, realiza tres mejoras sustanciales a partir de *Basic*.

1. **Filtrado de ancestros agregados a las transacciones**. No es necesario agregar a todos los ancestros de los items en una transacción. Como mejora se pueden agregar sólo a los ancestros que están en al menos uno de los itemset candidatos que están siendo considerados en la iteración actual.
2. **Pre-computar ancestros**. En vez de encontrar los ancestros de cada ítem recorriendo el grafo de la taxonomía se pueden pre-computar los ancestros de cada ítem.
3. **Eliminar itemsets que solo contienen un ítem y sus ancestros**. Para justificar esta optimización se presentan dos lemas:

   Lemma 1. *El soporte de un itemset X que contiene un ítem x y su ancestro $\hat{x}$ tiene el mismo soporte que el itemset $X - \hat{x}$*

   Lemma 2. *Si L_k, el set de itemset frecuentes, no contiene ningún itemset compuesto por un ítem y su ancestro, el conjunto de candidatos C_{k+1} generado por el procedimiento de generación de candidatos no incluirá ningún itemset que contenga un ítem y su ancestro.*

El **Lema 1** demuestra que no se necesita contar itemsets que solo contienen un ítem y su ancestro mientras que el **Lema 2** garantiza que eliminar este tipo de candidatos no va a generar nuevos candidatos con sólo items y sus ancestros en pasadas subsiguientes del algoritmo.

Veamos el algoritmo en pseudo-código:

Algorithm 4 Cumulate

Input *minsupp*, *Grafo dirigido sin ciclos G*, *Conjunto de transacciones D*

Computar G^* *de* G ▷ Optimización 2
$k \leftarrow 1$
$L_1 \leftarrow \{i \in \mathcal{I} \mid soporte(i) \geq minsupp\}$
$k \leftarrow 2$
while $L_k \neq \emptyset$ **do**
    $C_k \leftarrow$ Generar candidatos de tamaño k a partir de L_k
    if $k = 2$ **then**
        Eliminar cualquier candidato C_2 que
consista de un ítem y su ancestro ▷ Optimización 3
    end if
    Eliminar ancestros en G^* que no estén presentes
en ningún candidato C_k ▷ Optimización 1
    for $t \in D$ **do**
        for $x \in t$ **do**
            Agregar ancestros de x en G^* a t.
        end for
        for $c \in C_k$ **do**
            if $c \subseteq t$ **then**
                $\sigma(c) \leftarrow \sigma(c) + 1$
            end if
        end for
    end for
    $L_k \leftarrow \{c \in C_k \mid soporte(c) \geq minsupp\}$
    $k \leftarrow k + 1$
end while
return $\bigcup_k L_k$

Fuente:[Srikant and Agrawal, 1995]

La diferencia entre la generación de candidatos de *Apriori* y *Cumulate* es que en el último se hace un *pruning* especial en C_2 utilizando el Lemma 2.
La generación de reglas en *Cumulate* es idéntica a la de *Apriori* salvo por el filtrado de reglas *R-interesantes* al final del algoritmo, es decir , dado $Z = X \cup Y$. La regla $X \Rightarrow Y$ se mantiene si supera la confianza mínima y su soporte es R veces mayor o igual que el soporte esperado.

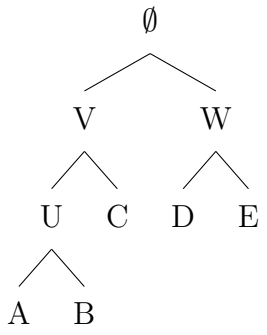
3.3. Modificaciones agregadas

Basándonos en el estudio y optimización previo del algoritmo *Apriori* se reutilizaron varias técnicas para optimizar *Cumulate*. Las modificaciones agregadas fueron verticalizar la base de datos (apartado 2.4.1), paralelización en la etapa de conteo de candidatos (apartado 2.4.2)

y paralelización en la generación de reglas (apartado 2.4.3). Estas implementaciones tienen el nombre de *Vertical Cumulate* y *Vertical Cumulate con paralelización*.

3.3.1. Expansión de las transacciones en la base vertical

Para aplicar la técnica de verticalizar la base de datos se tuvo que alterar el proceso para incorporar la taxonomías de los ítems. Una vez que se completa la estructura de datos que representa la base vertical se debe expandir agregando los ancestros de los ítems. Esto se puede realizar de forma sencilla recorriendo la taxonomía y agregando al tidset de cada ancestro los tidsets de sus hijos. Tomando el ejemplo de la base de datos ilustrada en el Cuadro 9, expandimos la base vertical con la taxonomía G:



Cuadro 10: Taxonomía

Items	Tidsets
A	1,2,4
B	2,3
C	1,2,3,4
D	1,3,4
E	1,3
U	1,2,3,4
V	1,2,3,4
W	1,3,4

Cuadro 11: Base Vertical Expandida

3.3.2. Cálculo de la esperanza de un ancestro con base vertical

Para implementar el concepto de R-interesante en el algoritmo *Rule-Generation* se debe calcular la esperanza del soporte basado en su ancestro. El calculo de $E_{\hat{Z}}[Pr(Z)]$ visto en la sección 3.1 requiere calcular el soporte del itemset ancestro $\hat{Z}$. En una representación de la base de datos horizontal, el cálculo del soporte de $\hat{Z}$ implica hacer un escaneo por las transacciones e ir verificando si $\hat{Z}$ está contenido en cada transacción. Al tener la base de datos vertical, se obtiene una gran ventaja para cuando se generan las reglas ya que no se debe escanear la base de datos nuevamente.

4. Algoritmo *HTAR*

4.1. Trabajos previos relacionados

Las reglas temporales buscan agregar al tiempo como una nueva variable esencial en cada transacción, utilizarlo para definir el nivel de detalle de la base de datos (granularidad) y así obtener los itemsets frecuentes en distintos períodos, sin descartarlos en caso de no cumplir con la frecuencia deseada en la totalidad de la base de datos. Existen múltiples trabajos que analizan la temporalidad de los datos para obtener patrones o sus regularidades, ya sea buscando asociaciones secuenciales [Agrawal and Srikant, 1995], cíclicas [Ozden et al., 1998], utilizando calendarios [Li et al., 2003], teniendo en cuenta los períodos de exhibición de los artículos [Chang et al., 2002] o sin tenerlos en cuenta [Ale and Rossi, 2000].

En esta ocasión vamos a focalizarnos en la implementación de un algoritmo de reglas de asociación basado en jerarquía granular temporal [Hong et al., 2016]. Esta aproximación toma como parámetro una jerarquía temporal, permitiéndole dividir a la base de datos en distintos *períodos* de interés.

4.2. Gránulos y Jerarquía Temporales

Definimos como *gránulos temporales* a cada intervalo de tiempo de interés que se utiliza para dividir la base de datos. Cada gránulo temporal tendrá, naturalmente, sus propios itemsets frecuentes y reglas de asociación válidas en ese período. Los gránulos temporales pueden unirse para formar nuevos gránulos más genéricos. Decimos que un gránulo de tiempo pj es *hijo* de otro gránulo pg si el período de tiempo de pj está contenido en pg. Cada relación padre-hijo constituye un *nivel*.

De esta manera se genera una jerarquía a la que llamaremos *Jerarquía Temporal*.

En este trabajo vamos a utilizar una jerarquía dividida en 4 niveles: *Quincena, Mes, Trimestre y Año*, tal como muestra la Figura 2. Está constituida por 4 niveles (0, 1, 2 y 3). Los mismos poseen 24, 12, 4 y 1 gránulos respectivamente. Cada gránulo es nombrado como *númeroDeNivel-númeroDePeríodo*. Así por ejemplo la primer quincena de Marzo (5Q) será el gránulo $0-5$, Agosto (AGO) será el gránulo $1-8$ y el 4to trimestre (4T) será el gránulo $2-4$, etc.

Los gránulos de tiempo de nivel 0 son llamados *Hojas* y poseen propiedades especiales a tratar más adelante.

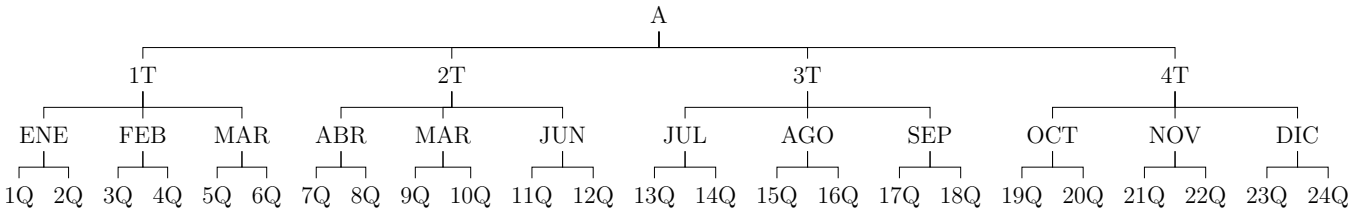


Figura 2: Jerarquía Temporal utilizada

Podemos hacer las siguientes observaciones

- Una jerarquía es válida si cada transacción y su respectivo timestamp pertenecen a uno y sólo un gránulo de tiempo por nivel, para todos los niveles.
- Todas las transacciones utilizadas para evaluaciones en este trabajo deben pertenecer al mismo año.
- El trabajo original permite al usuario establecer su propia jerarquía temporal.
- No es un requerimiento dividir en períodos iguales.

4.3. Detalle del algoritmo

4.3.1. Descripción

Así como el algoritmo Apriori, este algoritmo toma como parámetros una base de datos (con timestamps asociados a las transacciones), un soporte mínimo y una confianza mínima.

Primero se procede a identificar los itemsets frecuentes en cada gránulo temporal *hoja*, es decir, los que superen el soporte mínimo. Una vez obtenidos los itemsets frecuentes de cada hoja, se sube de nivel. Para cada gránulo del mismo se toman todos los itemsets frecuentes de los gránulos hijos que lo conforman, y luego se filtran evaluando su soporte ahora teniendo en cuenta la totalidad de transacciones del intervalo del gránulo padre. Este proceso se repite hasta llegar al último gránulo del nivel.

Una vez obtenidos los itemsets frecuentes de cada gránulo de cada nivel, se procede a generar las reglas de asociación que cumplan la confianza requerida, teniendo en cuenta las transacciones dentro del período de cada gránulo.

4.3.2. Correctitud y Completitud

Para evidenciar la correctitud y completitud del algoritmo, es necesario observar y demostrar los siguientes enunciados

A) **Si un itemset es frecuente en el periodo pg entonces es frecuente en algún período hijo pj que lo constituye.**

El problema puede reducirse de la siguiente manera:

Dado un set de transacciones $\mathcal{D}$ (nuestro gránulo pg), decimos que X (nuestro itemset) es frecuente respecto a un soporte mínimo s si

$$Soporte(X)_{\mathcal{D}} = \frac{Count(X)_{\mathcal{D}}}{|\mathcal{D}|} \geq s$$

Luego, si X es frecuente en $\mathcal{D}$, no existe partición $P = [d_1, d_2, ..d_n]$ de $\mathcal{D}$ tal que

$$Soporte(X)_{d_i} < s, \forall d_i \in P, |d_i| > 0$$

siendo $|d_i|$ la cantidad total de transacciones en el subconjunto d_i.

Por la misma lógica temporal, las transacciones de cada d_i corresponden a un mismo gránulo hijo de pg.

Dem. Por absurdo. Supongo que existe dicha partición P.

Entonces

$$Soporte(X)_{d_i} < s, \forall d_i \in P$$

$$\frac{Count(X)_{d_i}}{|d_i|} < s$$

$$Count(X)_{d_i} < s|d_i|$$

$$\sum_{i=1}^{n} Count(X)_{d_i} < s \sum_{i=1}^{n} |d_i|$$

$$\sum_{i=1}^{n} \frac{Count(X)_{d_i}}{|d_i|} < s$$

$$\frac{Count(X)_T}{|T|} < s$$

$$Soporte(X)_T < s$$

Pero habíamos dicho que X era frecuente en T.

Abs.■

De esto se concluye que no pueden aparecer nuevos itemsets frecuentes en un gránulo padre que no haya sido frecuente en alguno de sus hijos.

B) **Los itemsets frecuentes y reglas asociadas para una base de datos con el algoritmo Apriori deben ser los mismos que para el último nivel del HTAR**

Asumiendo que los dos algoritmos utilizan el mismo soporte y confianza mínimos, en ambos casos las reglas contemplan la totalidad de la base de datos, por lo que los resultados deberían ser *iguales*. Sin embargo, es necesario que todas las transacciones estén incluídas en un único año. De no estarlo, nuestra jerarquía temporal no sería válida ya que existirían transacciones que no pertenezcan a por lo menos un gránulo por nivel.

El algoritmo puede dividirse en 3 fases:

4.3.3. Fase 1: Obtener los itemset frecuentes temporales en los gránulos hoja

Siguiendo la metodología *Bottom-up*, lo primero que vamos a hacer es obtener los itemsets frecuentes temporales TFI_j para cada *gránulo hoja* p_j.

Algorithm 5 Finding Frequent Temporal Itemsets in leaf granules

Input *minsupp* s, *Set de transacciones* TDB_j *en el gránulo temporal pj*

1: $r \leftarrow 0$
2: $C_{jr} \leftarrow$ *todos los items en pj*
3: **while** $C_{jr} \neq \emptyset$ **do**
4: $\quad r \leftarrow r + 1$
5: $\quad TFI_{jr} \leftarrow \emptyset$
6: $\quad$ **for** $i \in C_{jr}$ **do**
7: $\quad\quad$ **if** $Sup(i)_{pj} > s$ **then**
8: $\quad\quad\quad TFI_{jr}$.agregar(i)
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad C_{jr} \leftarrow$ Generar todos los candidatos temporales de tamaño $r + 1$ de TFI_{jr}
12: **end while**
13: **return** $\bigcup_{k=1}^{k=r} TFI_{jk}$

Fuente: [Hong et al., 2016]

4.3.4. Fase 2: Obtener los itemset frecuentes jerárquico-temporales

Una vez obtenidos los itemset temporales frecuentes de cada hoja, subimos de nivel. Allí, para cada gránulo, se juntan los itemset frecuentes de los gránulos que lo integran y se filtran nuevamente, permanenciendo sólo los que cumplan el soporte, esta vez, en todo el período del gránulo padre.

Algorithm 6 Finding Hierarchical Temporal Frequent Itemsets

Input *minsup* s, *Conjunto de gránulos de tiempo jerárquico* HTG

for $pg \in HTG$ **do**
  $r \leftarrow 0$
  $PI_{pg} \leftarrow \bigcup TFI_j \in pg$
  for $X \in PI_{pg}$ **do**
    if $Sup(X)_{pg} <= s$ **then**
      $PI_{pg} \leftarrow PI_{pg} - X$
    end if
  end for
  $HTFI_{pg} \leftarrow PI_{pg}$
end for
return $\bigcup HTFI_{pg}$

Fuente: [Hong et al., 2016]

Junto con los itemsets frecuentes de los gránulos hoja de la fase anterior, obtenemos finalmente el conjunto de los itemsets frecuentes para cada gránulo de nuestra jerarquía al que denominaremos *HTFI*.

4.3.5. Fase 3: Generar las reglas jerárquico-temporales

Por úlitmo, para cada itemset frecuente de cada gránulo temporal, generamos todas las reglas de asociación y filtramos aquellas que no cumplan con la confianza mínima pedida. Evaluamos teniendo en cuenta el soporte y cantidad de transacciones totales relativas al gránulo correspondiente.

El algoritmo resultante es idéntico al *Rule-Generation* del algoritmo apriori, esta vez aplicado a cada gránulo.

Algorithm 7 Rule-Generation

Input *Conj. itemsets frecuentes por granulo* $HTFI$, *Conj. granulos* HTG , *minconf* c

for $pg \in HTG$ **do**
  $HTAR_{pg} \leftarrow \emptyset$
  for $X \in HTFI_{pg}$ **do**
    for $A \in subset(X)$ **do**
      $B \leftarrow X - A$
      if $Conf(A \Rightarrow B) \geq c$ **then**
        $HTAR_{pg} \leftarrow HTAR_{pg} \bigcup \{(A \Rightarrow B)\}$
      end if
    end for
  end for
end for
return $\bigcup HTAR_{pg}$ ▷ Observación 1

Fuente: [Hong et al., 2016]

Observación 1: La unión de reglas frecuentes están indexadas por gránulo temporal al cual pertenecen

4.4. Modificaciones Agregadas

4.4.1. MTP/MCP

MTP es la sigla utilizada para definir al *Período de tiempo Maximal* de un ítem. Esto es, el período de tiempo de la primer ocurrencia del item, hasta el último disponible en la base de datos. El MCP de un itemset es el *Período de tiempo Común Maximal* entre los MTP de los items que conforman el itemset. Formalmente, Sea la t_a el primer tiempo de aparición del item A pertenciente al período temporal q_i, decimos que $MTP(A) = (q_i, q_{i+1}, ...q_n)$ donde q_n es el último período de la base de datos.

Por consecuente, $MCP(A, B) = MTP(A) \cap MTP(B)$

Estos conceptos fueron introducidos con el objetivo de evitar la mayor cantidad de lecturas en disco posibles, leyendo sólo las transacciones pertinentes a los períodos de interés para así calcular el soporte relativo para cada itemset dado un período de tiempo.

En la implementación de este trabajo, nuestra base está verticalizada. Al leer la base una única vez para obtener el soporte relativo de un itemset i dado un período de tiempo pg, sólo hay que filtrar los id de las transacciones que pertenecen al período granular deseado y tomar su cantidad. Luego éste se divide por la cantidad total de transacciones en dicho período y se obtiene el soporte deseado.

4.4.2. FAP/LAP

El FAP o *First Appearance Period* en el paper original es una tabla que almacena, para cada elemento, el período de tiempo correspondiente a la primera transacción en la que aparece. El LAP es su análogo pero para el último período de aparición, propuesto en la presentación de esta tesis como posible mejora, dado que de no tenerlo, se consideraba que el MTP de un ítem terminaba en el último período de la base de datos.

En la propuesta se pensó utilizar el FAP y LAP para precalcular el MCP de un itemset. Teniendo el período de primera aparición de cada ítem en el itemset, sólo restaba tomar los períodos pertenecientes al intervalo común mayor de coincidencia, para luego filtrar por aquellos períodos que están contenidos en el gránulo temporal deseado. Luego se obtendrá su soporte relativo haciendo un nueva lectura de la base de datos, pero esta vez teniendo en cuenta sólo las transacciones pertenecientes a períodos que pertenezcan a este intervalo.

En la implementación de esta tesis sin embargo, para obtener el soporte relativo de un itemset, primero filtramos los id de transacción que pertenezcan al período deseado para cada ítem i. Luego buscamos la intersección de estos, y finalmente dividimos por la cantidad total de transacciones en ese período para obtener el soporte final. Si bien necesitamos saber los límites de los períodos a evaluar (esta vez en términos de id de transacción) para poder hacer un correcto filtro a cada ítem i, podemos prescindir de las tablas FAP y LAP.

4.4.3. El parámetro lambda

El paper original hace referencia a un parámetro λ que nunca menciona de dónde se obtiene. Éste se utiliza como *soporte mínimo* para el cálculo de itemsets frecuentes **sólo para los gránulos hojas** de nuestra jerarquía.

Para garantizar la correctitud y completitud de este algoritmo, es necesario establecer el parámetro λ **igual** que el soporte mínimo pasado como parámetro. Utilizar un valor distinto dependiendo del nivel del gránulo temporal analizado afecta seriamente los resultados y tiempos obtenidos.

Por ejemplo, si tomamos un parámetro λ mayor que el soporte mínimo, estaremos endureciendo nuestro concepto de itemset frecuente en los gránulos hoja, en la fase 1 del algoritmo. Por consiguiente, al empezar la fase 2, partiríamos de una base mucho mas reducida de items, generando aún menos itemsets frecuentes en los gránulos no-hoja y así también menos reglas.

En este trabajo tomaremos por predeterminado el valor λ igual al soporte mínimo, aunque en la publicación de la librería este parámetro está disponible para ser alterado.

4.4.4. Modo de cálculo de soporte relativo

Dado un gránulo temporal, para saber si un itemset es frecuente es necesario tomar su soporte y dividirlo por la cantidad total de transacciones en dicho gránulo.

La manera que tiene el paper original de lograr esto es haciendo lecturas en disco para cada ítem perteneciente al itemset (ahorrándose las lecturas de períodos donde no pertenece por medio de FAP/LAP).

En nuestro caso, al poseer la base de datos verticalizada, basta con buscar la intersección de los id de transacciones (tid) pertenecientes al gránulo temporal en las cuales aparecen todos los ítems del itemset.

Para lograr esto, se debe previamente filtrar las transacciones de cada ítem y quedarse sólo con aquellas que pertenezcan al gránulo deseado. Teniendo previamente almacenados el id de la transacción que inicia y el id de la transacción que termina cada gránulo (lb, ub), se proponen dos maneras de hacerlo:

- Empezar por el comienzo del arreglo. Recorrer e ignorar todos los tid mientras no se llegue al inicio del gránulo lb. Una vez alcanzado, recorrer y acumular el resto de los tid hasta encontrar un valor igual o superior al final del gránulo ub, o finalizar el arreglo. Llamaraemos a este modo **Lineal**.
- Igual que el modo lineal, excepto que en vez de comenzar desde el principio del arreglo, implementamos una *búsqueda binaria* para encontrar el primer tid igual o mayor que el inicio del gránulo lb. Luego se recorre normalmente. Llamaremos a este modo **Binario**.

Recordemos que las transacciones poseen un sistema de numeración secuencial que respeta el orden temporal. Esto quiere decir que si el id de la transacción A es menor al id de la transacción de B, podemos concluir que A ocurrió antes de B (posee un timestamp menor)

Veamos un ejemplo en la Figura 3. Queremos ver si el itemset (manzana, uva) es frecuente en el mes de Marzo. Supongamos que las transacciones de este mes están incluidas en el intervalo [98, 310](lb, ub), o sea, la primer transacción de marzo tiene id igual a 98 y la última 310.

Los arreglos de la figura muestran todas las transacciones en las cuales la manzana y la uva respectivamente aparecen en nuestra base de datos. Podemos ver que ambos items poseen 4 transacciones en común.

1	5	8	10	11	19	20	77	80	102	**110**	**112**	**150**	214	**273**	300	720

8	77	80	**110**	111	**112**	113	140	**150**	170	200	**273**	720

Figura 3: Ejemplo cálculo de soporte relativo

Para encontrar estas transacciones se debe proceder de la forma anteriormente descripta. Observar cómo de manera **Lineal** en el caso de la manzana es necesario recorrer 10 transacciones antes de encontrar la primera perteneciente al mes de marzo. Estos tiempos se reducirían de manera significativa con la modalidad **Binaria**, evidenciándose aún más a medida que crezca la cantidad de apariciones de un mismo ítem en nuestra base de datos.

La eficiencia de estos modos será verificada en la sección de experimentación.

4.4.5. Paralelización del cálculo del soporte de candidatos

En la seccion 2.4.2 hemos visto la paralelización en la generación de candidatos de largo k. Esta forma de paralelización es utilizada tanto en *Apriori* como en *Vertical Cumulate*. En este caso sin embargo, implementaremos una paralelización no en los elementos de largo k, sino en los **gránulos temporales por nivel**. Hemos visto que para cada nivel, es necesario calcular los itemsets frecuentes del granulo (en caso de los gránulos hoja) o los frecuentes de las hojas que lo integran (en el caso de los gránulos no-hoja).

Esto es posible gracias a que, para un mismo nivel, estas tareas son paralelizables ya que se puede prescindir del resultados de los gránulos hermanos del ejecutado, no así al avanzar de nivel.

Por ejemplo, es perfectamente paralelizable el cálculo de itemsets frecuentes en la primer quincena de Enero y la segunda quincena de Marzo pero no así entre la primer quincena de Enero y el primer trimestre, ya que este último requiere un preprocesamiento de los gránulos hijos/hojas que lo integran.

La eficiencia de este tipo de paralelización, junto con la paralelización de elementos para un mismo k y el algoritmo sin paralelización será verificada en la sección de experimentación.

4.4.6. Paralelización del cálculo de generación de reglas

De la misma manera que la sección anterior, la generación de reglas es paralelizable por gránulo temporal, incluso sin diferenciar por nivel. Esto se debe a que los itemsets frecuentes para cada gránulo ya fueron calculados previamente, y las reglas generadas en cada uno no afectan de ninguna manera al resto de reglas generadas en otros gránulos. Cada gránulo necesita sólo su propio diccionario de itemsets frecuentes con sus respectivos soportes para la generación de reglas.

La eficiencia de esta paralelización será verificada en la sección de experimentación.

5. Algoritmo HTGAR

Este algoritmo es una combinación de *Cumulate* y *HTAR*. El objetivo es implementar una versión de *HTAR* que incorpore el concepto de jerarquías y las optimizaciones de *Cumulate* y devolver reglas generalizadas jerárquico-temporales (sección 1.3).

Para poder generar reglas generalizadas en el algoritmo *HTAR* se va a utilizar la estrategia de verticalización de la base por lo tanto se va a realizar una expansión de cada transacción para armar la base vertical al igual que *Cumulate* (sección 3.3.1).

Una vez armada la base vertical se va a ejecutar *HTAR* con una pequeña modificación en la sección de generación de candidatos. Se va a aplicar la optimización de *Cumulate* (sección 3.2) donde se realiza un pruning cuando $k = 2$ eliminando itemsets que estén compuestos por un ítem y su ancestro.

La generación de itemset frecuentes y reglas es igual que en *HTAR* solo que las reglas de asociación generadas tienen la característica de ser generalizadas por la incorporación de taxonomía. Por ende, para cada regla generada se le puede atribuir:

- Uno o varios gránulos temporales donde esta regla es frecuente.
- Para cada ítem que integra la regla, un nodo de la taxonomía

Veamos un ejemplo con el cuadro 7. Supongamos que obtenemos la regla $\{C\} \Rightarrow \{W\}$ con un soporte y confianza mínima en 50 %. Viendo el cuadro 6 observamos que C es una hoja de la taxonomía mientras que W es una raíz. Por otro lado observando la Figura 2, la regla ocurre en los periodos: 8Q, ABR y 2T.

Al algoritmo lo llamaremos *Hierarchy Temporal Generalized Association Rules* o en su forma breve: **HTGAR**.

6. Experimentación

En esta sección se harán experimentos con todos los algoritmos implementados. Los objetivos de la experimentación en cada algoritmo se detallan a continuación:

- ***Apriori***. Se buscará comparar la performance del algoritmo con y sin paralelismo.
- ***Cumulate***. Se buscará analizar la performance del algoritmo comparando tres versiones: sin optimizaciones, con optimización de base vertical, y con optimización de base vertical y paralelismo. El objetivo es experimentar con distintos datasets que varían aspectos de la taxonomía. También se evaluará la efectividad de la medida R-interesante.
- ***HTAR***. Se buscará analizar de manera cuantitativa los itemsets candidatos, frecuentes y las reglas generalizadas obtenidas al aplicar el algoritmo a una base de datos real. Luego, con un dataset sintético, se comparará la eficiencia de la paralelización de tareas al generar los itemset frecuentes y la creación de reglas, cada uno con su respectivo experimento. También se buscará evidenciar la ganancia obtenida al modificar el método de cálculo del soporte relativo.
- ***HTGAR***. Se buscará analizar si la performance es similar a *Cumulate* bajo un experimento que varía un aspecto de la taxonomía de los datos. También se hará un análisis cualitativo de la medida R-interesante frente al tipo de reglas generalizadas jerárquico-temporales.

Los experimentos fueron ejecutados en Pypy 3.11, una implementación alternativa más rápida del lenguaje Python. Para la implementación del proceso en paralelo se utilizó la librería de Python *multiprocessing* con la configuración que viene por defecto que establece la cantidad de procesos a correr en paralelo como la cantidad de CPUs de la maquina.

Los experimentos de la sección *Apriori*, *Cumulate* y el experimento 6.5.1 de la sección *HTGAR* fueron realizados en una computadora con procesador Intel i7-9700K de 3.60GHz y 16GB RAM con sistema operativo Windows 10. Los experimentos de la sección *HTAR* y el experimento 6.5.2 de la sección *HTGAR* fueron realizados en una computadora con procesador Intel i7-1165G7 de 4.7GHz y 16GB RAM con sistema operativo Ubuntu 20.04.

Si bien el tiempo representado en los experimentos corresponde a una única ejecución, los mismos fueron ejecutados varias veces obteniéndose resultados similares. Además, las computadoras utilizadas para la experimentación previamente descriptas se utilizaron de manera dedicada.

6.1. Conjunto de datos

6.1.1. Datos reales

Los datos reales fueron tomados de un dataset llamado *Foodmart* que viene con el paquete de software Microsoft SQL Server 2000. Este dataset también fue utilizado en [Hong et al., 2016] para realizar experimentos. Incluye dos años de ventas de productos de supermercados, con un

total de 54.537 transacciones, 1.559 productos y un promedio de $4,6$ items por transacción. La taxonomía tiene 5 niveles y 3 raíces.

Debido a la distribución de los datos de este dataset, es necesario bajar mucho el soporte para obtener itemsets frecuentes, lo que ocasiona una volatilidad de la cantidad de los mismos.[1].

Teniendo en cuenta esto, *Foodmart* se utilizó principalmente para obtener métricas genéricas y destacar cualidades significativas del algoritmo HTAR en comparación al Apriori base.

6.1.2. Datos sintéticos

Los datos sintéticos fueron generados por el programa *IBM Quest Synthetic Data Generator* utilizado en [Agrawal and Srikant, 1994] y [Srikant and Agrawal, 1995]. El programa se obtuvo del repositorio `https://github.com/nrthyrk/quest` [2]. En el Cuadro 12 se muestran los distintos parámetros por defecto del generador que se utilizaron para poder comparar resultados.

Una cosa destacable a mencionar es que el generador de datos sintéticos no posee la opción de agregar timestamps a las transacciones. Para experimentar con los algoritmos que implementan reglas de asociación **temporales** los datasets requieren una asociación de tiempo a cada transacción. En el paper [Hong et al., 2016] asignan timestamps de forma aleatoria a las transacciones (y por lo tanto irreplicable). En nuestro caso, se decidió simular la misma distribución que posee el dataset de *Foodmart*. Esto se logró asignando el proporcional de transacciones en cada día respecto a esa misma cantidad en el dataset de datos reales. Esta forma soporta el aumento de la cantidad de transacciones de manera arbitraria, lo que es necesario para la parte de experimentación.

Parámetros		Valores por defecto
$\|\mathcal{D}\|$	Numero de transacciones	1.000.000
$\|T\|$	Largo promedio de una transacción	10
$\|I\|$	Largo promedio del itemset frequent maximal	4
$\|\mathcal{I}\|$	Cantidad de itemsets frequentes maximales	10.000
N	Cantidad de items	100.000
R	Cantidad de raíces	250
L	Cantidad de niveles de la taxonomía	4
F	Cantidad de hijos por nodo interno	5
D	Ratio de profundidad	1

Cuadro 12: Parámetros del generador de datos sintéticos

$$\text{Ratio de profundidad} = \frac{\textit{probabilidad de que un ítem de una regla provenga del nivel } i}{\textit{probabilidad de que un ítem provenga del nivel } i+1}$$

[1] Para el caso de la experimentación con *HTAR*, el paper original utiliza *Foodmart* en sus experimentos pero sólo mide los tiempos de lectura que resultan estériles de comparar al haber sido implementados en una base horizontal, siendo nuestra implementación sobre una base vertical.

[2] Se hicieron algunas modificaciones al formato de salida de los datasets para que los puedan leer nuestros algoritmos.

6.2. Experimentación Apriori

6.2.1. Efectividad de la paralelización

Observamos la eficiencia de la paralelización del algoritmo *Apriori* en la etapa de conteo de candidatos y la generación de reglas.

Etapa conteo de candidatos.

Para la confección de la Figura 4, tomamos un dataset sintético de 1.000.000 de transacciones, 100.000 items y largo promedio de una transacción es de 10 items. Ejecutamos el algoritmo con soporte mínimo en $0,5\,\%$. La mayor cantidad de candidatos se generan en $k = 2$ y es en el único punto donde la paralelización demuestra una ganancia considerable en términos de performance temporal. Para verificar este hallazgo, analizamos la tarea a paralelizar: Esta consiste en calcular el soporte de un candidato que en la versión de base vertical es hacer una intersección entre las transacciones en las que aparece cada ítem del candidato (apartado 2.4.1).

Llegamos a la conclusión de que cuanto más candidatos hay en una iteración k, el tiempo de ejecución de esa iteración es mayor en la versión secuencial ya que este conteo se realiza en secuencia.

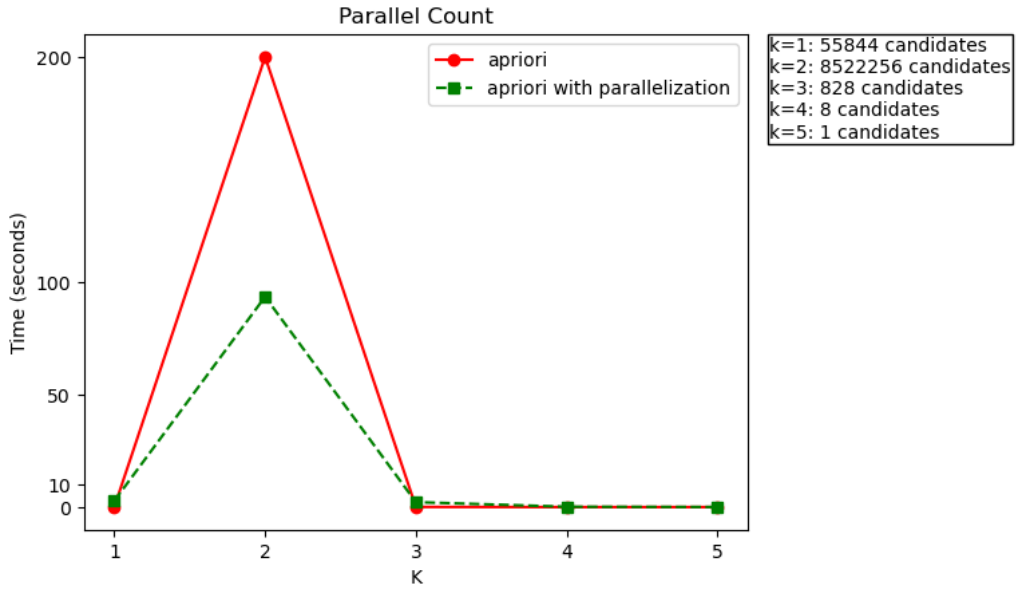


Figura 4: Tiempos de ejecución de Apriori con y sin paralelización en la etapa de conteo de candidatos

Etapa generación de reglas.

Para la confección de la Figura 5, aplicamos la paralelización en la etapa de generación de reglas experimentando con distintos tamaños de itemsets frecuentes. Para observar la efectividad de la

ejecución en paralelo expandimos los itemset frecuentes en conjuntos de reglas potenciales. Cada regla potencial que deriva de un conjunto frecuente tiene el antecedente y el consecuente. Por ejemplo si el itemset $\{A, B\}$ es frecuente entonces se puede expandir en $\{A \Rightarrow B\}$ y $\{B \Rightarrow A\}$. Luego la tarea a realizar en paralelo o secuencial es tomar una regla potencial $\{A \Rightarrow B\}$, calcular su confianza y verificar si es mayor que el umbral propuesto por el usuario.

A diferencia de la paralelización en la etapa de conteo de candidatos, en la generación de reglas no se observó una mejora de performance. Esto se debe a que la tarea a paralelizar es bastante simple en comparación a la tarea de calcular el soporte de un candidato ya que el soporte del antecedente y consecuente fueron calculados previamente en la etapa de generación de frecuentes. Con la estrategia de paralelización tomada, por mas que aumente el tamaño de la tarea paralelizable, hacerlo de forma secuencial es la mejor opción.

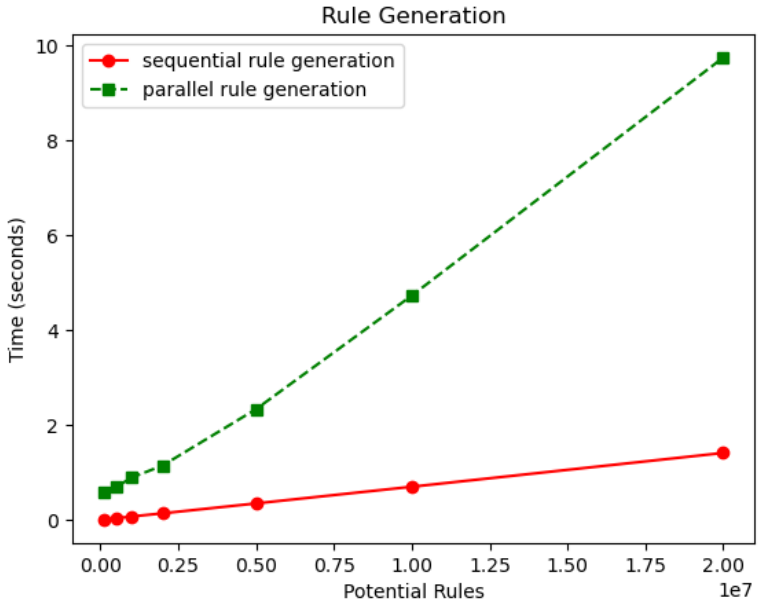


Figura 5: Tiempos de ejecución de la generación de reglas candidatas candidatas con y sin paralelización

6.3. Experimentación Cumulate

6.3.1. Comparación *Cumulate*, *Vertical Cumulate* y *Vertical Cumulate con paralelización*

La experimentación realizada en esta sección es la misma que en [Srikant and Agrawal, 1995] sólo que en ese paper comparan la performance de *Cumulate* con otros algoritmos que generan reglas de asociación generalizadas como *Basic* y *EstMerge*. Aquí se va a estar comparando la performance de *Cumulate*, *Vertical Cumulate* y *Vertical Cumulate con paralelización*.

Se realizaron 6 experimentos con datos sintéticos cambiando un parámetro en cada experimento dejando el resto en su valor por defecto como se muestra en el cuadro 12. El soporte mínimo se dejo en $0,5\,\%$ excepto para el experimento que varía el soporte.

El objetivo de estos experimentos es comparar la performance de los algoritmos implementados en diferentes situaciones. Primero se estudió la performance con diferentes características de la taxonomía. Estas son: cantidad de hijos por nodo, cantidad de raíces y ratio de profundidad. En segundo lugar se estudió la performance con distintas escalas de la base de datos. Estas son: cantidad de transacciones y cantidad de items. Por último se estudió la performance de los algoritmos variando los soportes mínimos. Los gráficos de los experimentos se encuentran en las Figuras 6, 7, 8.

Variación de hijos por nodo (Fanout). [6]

Se varió la cantidad de hijos por nodo entre 5 y 25. Esto genera que los niveles de la taxonomía decrementen a medida que las raíces aumentan ya que la cantidad total de items queda fijo en su valor por defecto como se muestra en el Cuadro 12. Vemos que las versiones con implementación de base de datos vertical son entre 3 y 6 veces mas rápidas que *Cumulate* para 5 hijos por nodos. Al incrementar el fanout el gap de performance decrece ya que al tener 25 hijos por nodo los niveles de la taxonomía son bajos y por ende hay menos ancestros. Esto implica que las expansiones de las transacciones son pocas teniendo menos candidatos para analizar.

Variación de raíces (Root). [6]

Se varió la cantidad de raíces entre 250 y 1.000. Esto tiene un efecto similar que en Fanout ya que a medida que la cantidad de las raíces se incrementa, la probabilidad de que una raíz específica esté en una transacción disminuye.

Variación del ratio de profundidad (Depth Ratio). [6]

Se varió el ratio de profundidad entre $0,5$ y 2. Con ratios más altos, los itemsets frecuentes serán elegidos de las hojas o niveles bajos de la taxonomía mientras que con un ratio bajo los items frecuentes provendrán de niveles más altos de la taxonomía. Si bien las versiones verticales superan a la implementación horizontal de *Cumulate*, la performance con paralelización no varía a medida que se aumenta el ratio.

Variación de transacciones. [7]

Se varió la cantidad de transacciones entre 100.000 y 10.000.000. En lugar de mostrar el tiempo de ejecución de cada algoritmo, se decidió mostrar el tiempo de ejecución dividido por el número de transacciones normalizado de tal forma que el tiempo tomado en cada algoritmo para 1 millón de transacciones es 1 unidad. Es decir, llamemos z al tiempo que tardó en ejecutarse cualquiera de los tres algoritmos para 1 millón de transacciones. Para el resto de las ejecuciones obtenemos los pares (x, y) donde x es la cantidad de transacciones e y es el tiempo de ejecución. La fórmula para normalizar cada y sería $y_{normalizado} = (y/x)/z$. En la Figura 7 se muestra que la implementación de *Cumulate* mantiene la relación en 1 a pesar del incremento de transacciones, lo cual se debe a que las operaciones siguen siendo las mismas sólo que es necesario escanear más transacciones. En cambio para las versiones verticales la relación no se mantiene en 1 ya que la estructura interna de la base vertical debe guardar tidsets mucho más grandes a medida que la cantidad de transacciones aumenta. Esto genera un costo en la operación de intersección de tids para calcular el soporte de los candidatos.
En el caso particular de *Vertical Cumulate con paralelización* con 100.000 transacciones ocurre que la relación entre el tiempo de ejecución y la cantidad de transacciones (normalizado) es mayor a 1. Es decir, el tiempo para calcular los itemsets frecuentes en un dataset chico es más alto de lo esperado. Al haber menor cantidad de transacciones, baja la cantidad de candidatos para contar y en ese caso la paralelización no es tan efectiva.

Variación de items. [7]

Se varió la cantidad de ítems entre 10.000 y 100.000. Esto genera que cambie la cantidad de niveles en la taxonomía. Con pocos items la taxonomía no crece en niveles, en cambio con muchos items la altura del árbol aumenta. Vemos que la performance del *Cumulate* empeora a medida que aumentan los items con respecto a las versiones verticales ya que si bien la cantidad de transacciones es constante, aumentar los items implica escanear más candidatos. Las versiones verticales mantienen un tiempo de ejecución similar a medida que los items aumentan. Esto se debe a que la estructura de la base vertical sólo crece en cantidad de items y no en el largo de los conjuntos de tids que tiene cada ítem.

Variación del soporte mínimo. [8]

Se varió el soporte mínimo entre 2 % y 0,3 %. Este experimento parece tener el mismo efecto que el de variar raíces o hijos por nodo ya que al tener un soporte bajo hay más itemsets frecuentes. Si el soporte aumenta, la cantidad de itemsets frecuentes baja en cada iteración. La diferencia que hay entre las versiones horizontales y verticales se debe a que la primera hace la lectura por todas las transacciones en cada iteración mientras que la implementación con base de datos vertical sólo paga el costo del conteo de candidatos.

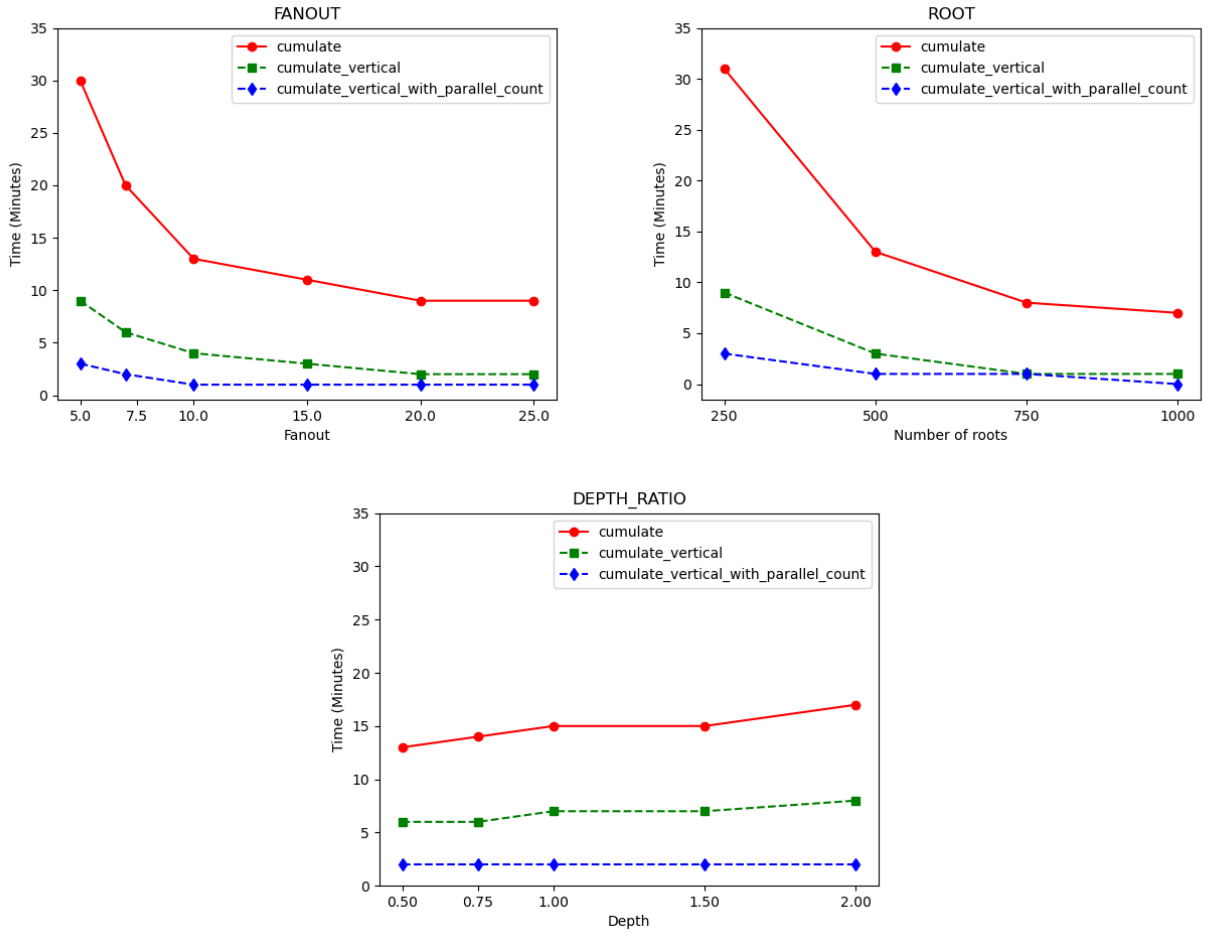


Figura 6: Tiempos de ejecución variando las escalas de taxonomía: Fanout, Roots y Depth Ratio.

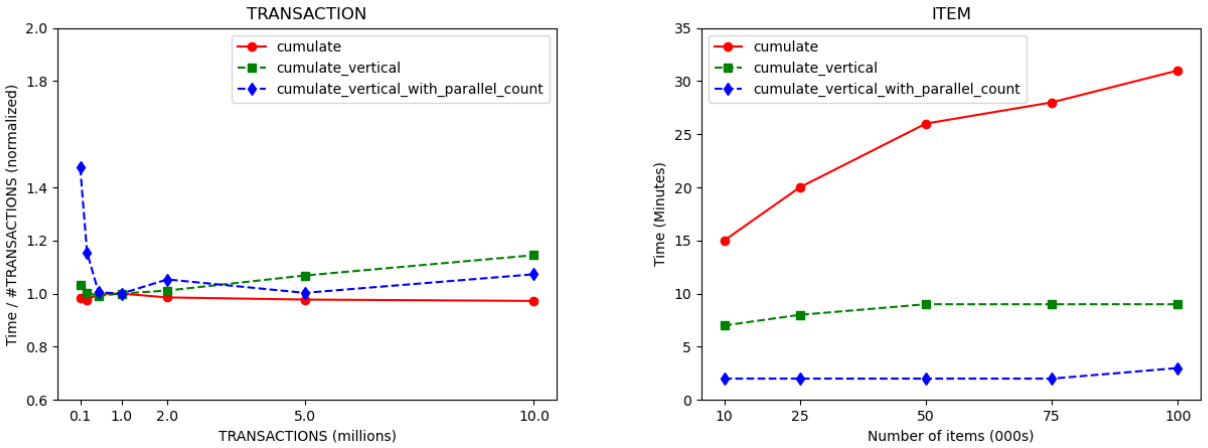


Figura 7: Tiempos de ejecución variando las escalas de la base de datos: Transacciones y Cantidad de items.

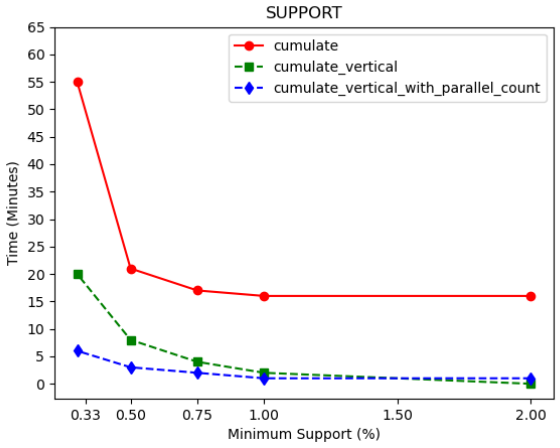


Figura 8: Tiempos de ejecución escalando el soporte.

6.3.2. Efectividad de la medida *R-interesante*

Observamos la efectividad de la medida *R-interesante* haciendo un pruning de reglas en el dataset con datos reales con una confianza mínima de 25 % y 50 % y soporte mínimo de 1 %. Estos parámetros son los mismos que se utilizaron en [Srikant and Agrawal, 1995] para la experimentación con el dataset real. Para la base de *Foodmart* entre un 35 % a un 42 % de las reglas fueron pruneadas con un nivel R-interesante mínimo de 1.1 para ambos niveles de confianza mencionados.

Por ejemplo la regla $\{BathroomProducts, Drink\} \Rightarrow \{CannedFoods\}$ fue pruneado porque su soporte y confianza fueron 1.1 veces menor que el soporte esperado del ancestro

$\{BathroomProducts, Drink\} \Rightarrow \{Foods\}$ donde *Foods* es un ancestro de *Canned Foods*.

6.4. Experimentación HTAR

6.4.1. Jerarquía temporal

Tal como se detalló en la sección 4.2, en todo momento utilizaremos como jerarquía temporal la expuesta en la figura 2, ya que fue elegida como jerarquía por default. A su vez, la jerarquía utilizada por el paper original [Hong et al., 2016] en los experimentos fue en tres niveles con 10, 5 y 1 gránulo temporal en cada uno respectivamente. Además de que no distingue exactamente cómo son generadas estas particiones, tampoco se encontraron itemsets frecuentes ni reglas interesante para la elaboración de la experimentación, es por eso que fue descartada.

6.4.2. Comparación *Apriori* con *HTAR*

En esta sección vamos a comparar de manera cuantitativa las reglas generadas por el algoritmo apriori original contra el algoritmo *HTAR*. Para esto se tomó la base **Foodmart'97**, **soporte = 0.02 %** y **confianza = 60 %**. Los parámetros seleccionados corresponden a los valores mínimos necesarios para obtener una cantidad de itemsets y reglas adecuados para la experimentación.

Propiedad	Cantidad
Itemsets candidatos evaluados	1.216.023
Itemsets frecuentes obtenidos	1.735
Reglas de asociación generadas	6

Cuadro 13: Resultados Apriori original

En las figuras 9, 10 y 11 se muestran los resultados del cuadro 13 para el algoritmo *HTAR*.

Cantidad de Itemsets Candidatos [9]

Cada hoja tiene una cantidad de candidatos equivalente al total del Apriori. Esto muestra que el dataset está bien distribuido, aunque se puede ver en el período $0-14$ una notoria concentración de items. Es posible que este período siga destacándose en las siguientes etapas del algoritmo.

Existe una drástica reducción de la cantidad de candidatos entre el nivel 0 y el nivel 1. Esto es debido a que al momento de salir de los nodos hoja, el nivel 1 utiliza como candidatos sólo los itemsets *frecuentes* de los períodos del nivel 0 que lo componen. Al bajar la cantidad de candidatos a evaluar, también mejoramos la performance del algoritmo. La correctitud de esta sección fue demostrada en 4.3.2.

Notar cómo el punto anterior aplica para todos los gránulos de tiempo que no sean hoja. Esto se debe a que el algoritmo establece que estos gránulos, deben siempre bajar hasta los nodos hoja que lo integren y obtener de allí la unión de sus itemsets frecuentes para utilizarlos como candidatos. Como consecuencia, no se pueden eliminar más candidatos en cada nivel, siendo las cantidades siguientes fácilmente calculables. Ejemplo: La cantidad de itemsets frecuentes del gránulo $2-4$ es la suma de los candidatos de los gránulos de nivel 1 que lo integran $(1-10,$

$1-11$ y $1-12$). Y estos a su vez son la suma de los itemsets frecuentes de los gránulos de nivel 0 que lo integran, que se pueden observar en el nivel 0 de la Figura 10.

Finalmente, para el nivel $3-1$, o sea la totalidad del año en la base de datos, se evalúan 2.4 millones de itemsets candidatos. El doble que en el algoritmo Apriori original. Esto es una gran desventaja que podría ser mejorada si se filtraran los candidatos a medida que se avanza de nivel.

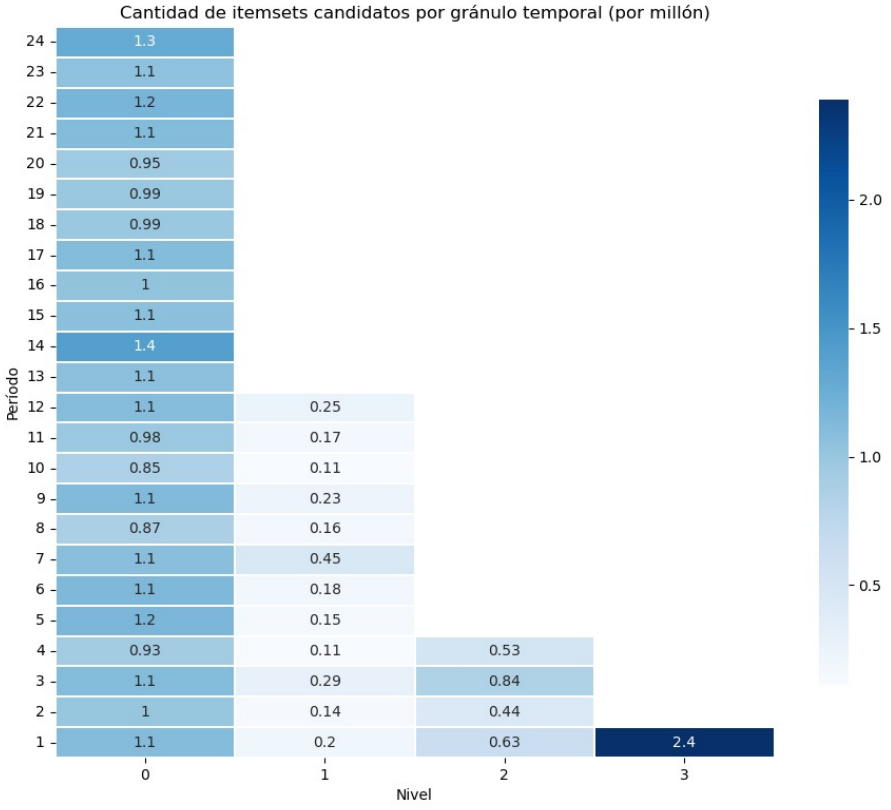


Figura 9: Heatmap de los itemsets candidatos por cada gránulo temporal

Cantidad de Itemsets Frecuentes [10]

El soporte utilizado es tan bajo que todos los itemsets frecuentes en el nivel 0 lo son también en el nivel 1. Ver que el nivel 1 posee los mismos valores en esta figura y la figura 9. Sin embargo para los próximos niveles, ya se depuraron la gran mayoría de estos y la cantidad baja drásticamente.

Se puede ver con mayor claridad un "período de oro" correspondiente al segundo trimestre $(2-2)$. Generado principalmente por el mes de Julio $(1-7)$ y en especial por su segunda

quincena $(0 - 14)$. Este trazo trasversal de los niveles de tiempo, permite la localización con mayor precisión de los períodos de injerencia en el cálculo del soporte de los itemsets frecuentes. Esto es uno de los mayores aportes del algoritmo.

Por supuesto, los motivos de este período de oro pertenecen al dominio del negocio y el origen de los datos que exceden a este trabajo.

Finalmente para el gránulo $3 - 1$, el algoritmo produce la misma cantidad de candidatos frecuentes que Apriori, como era de esperar.

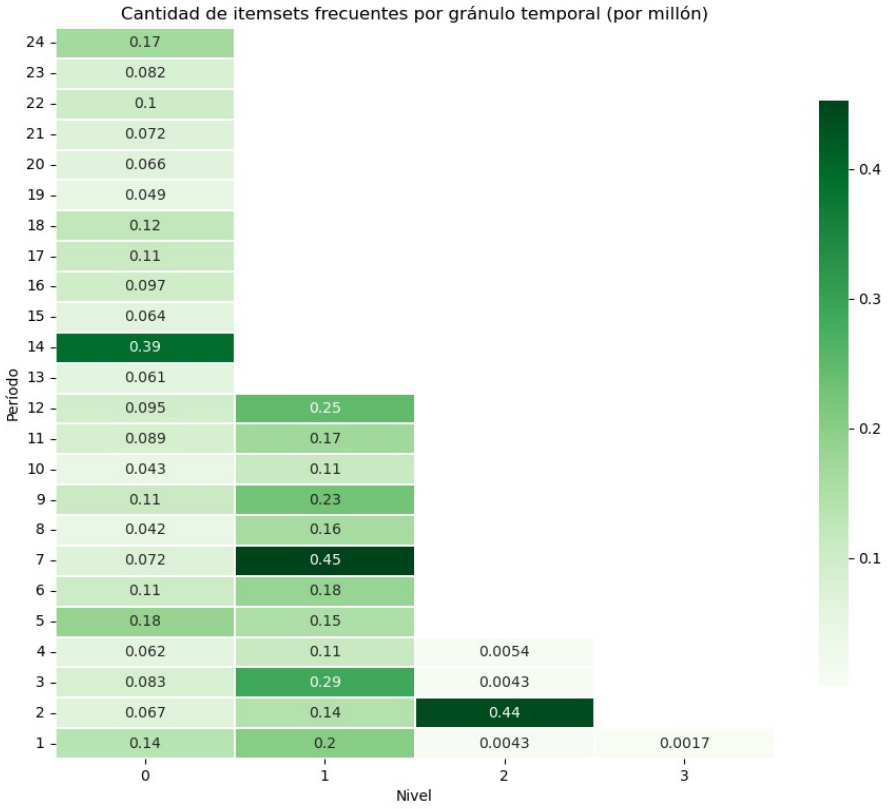


Figura 10: Heatmap de los itemsets frecuentes por cada gránulo temporal

Cantidad de Reglas generadas [11]

El gránulo de oro observado en la figura 10 posee su correspondencia con las reglas generadas. Los gránulos que forman parte de éste poseen mayor cantidad de reglas que el resto.

La confianza es tan baja que en niveles más bajos casi toda posible regla termina superando la confianza mínima. Recordar que siempre se tiene como referencia el soporte total del gránulo

correspondiente.

Mientras que el gránulo $3-1$ generó solo 6 reglas (como Apriori) en total se produjeron más de 30 millones. Esto podría conllevar problemas de memoria en la implementación final de la librería ya que implica un gran costo de almacenamiento.

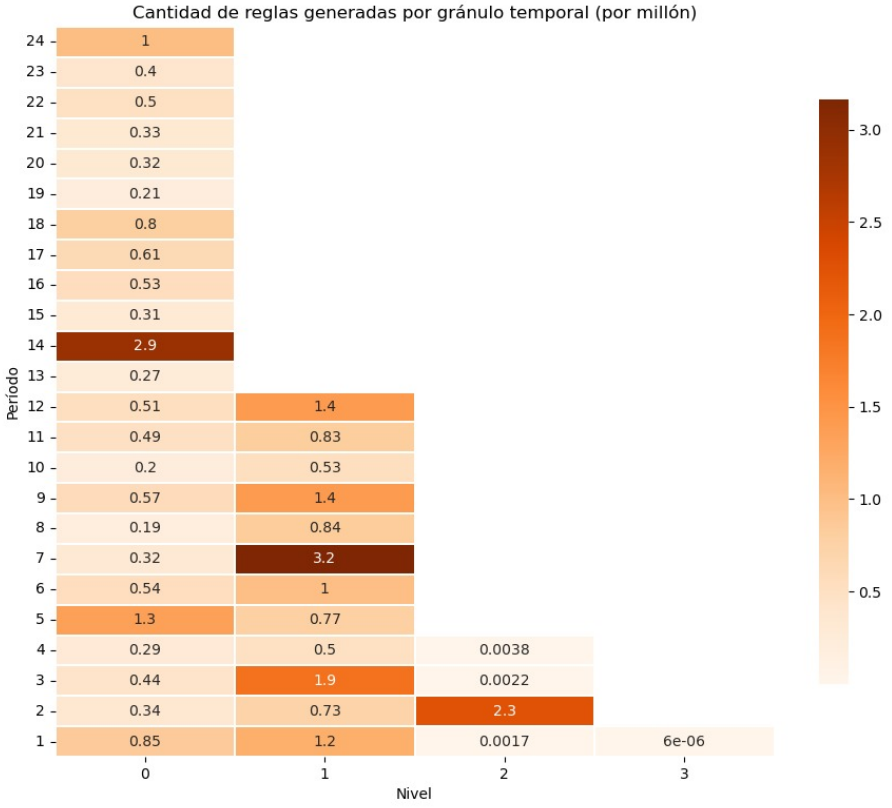


Figura 11: Heatmap de las reglas obtenidas por cada granulo temporal

6.4.3. Modo de cálculo de soporte relativo

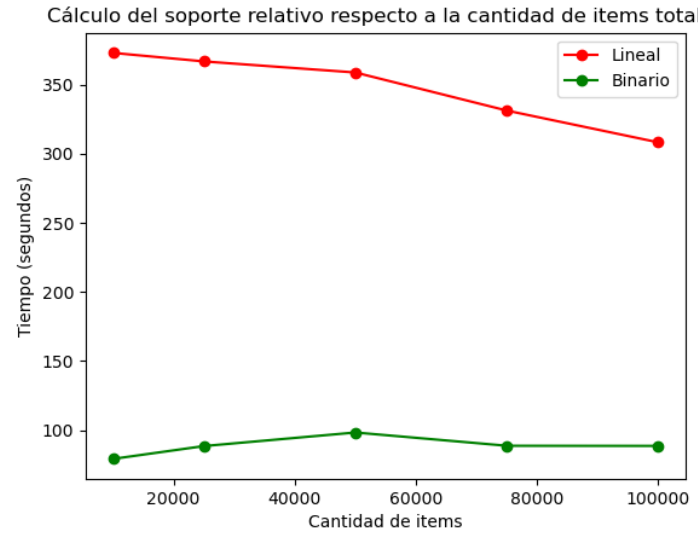
En la sección 4.4.4 introdujimos el concepto de Cálculo de soporte relativo Lineal y Binario. En los siguientes experimentos buscamos notar una diferencia significativa de performance entre ambos algoritmos. Al confeccionar la figura [12] podemos hacer las siguientes observaciones

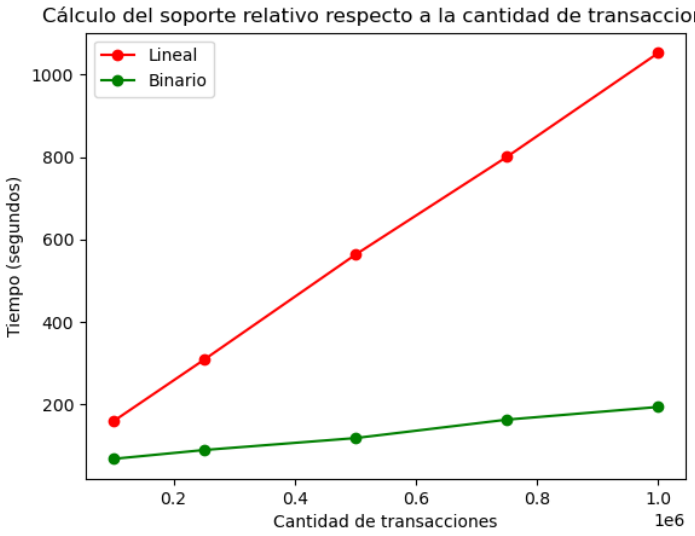
Al variar la cantidad de items podemos observar cómo la diferencia es muy amplia, aunque la complejidad de la forma lineal tiende a decrecer cuanto más elementos haya en el inventario total. Esto es esperable, ya que de haber más cantidad de elementos pero la misma cantidad de transacciones, hay menos probabilidad de que cada elemento aparezca en la transacción. Esto

genera que la cantidad de transacciones donde aparece cada uno es menor, lo que conduce a una menor comparación total de elementos al momento de buscar el soporte de manera lineal.

Variar la cantidad de transacciones corrobora nuestra hipótesis. Vemos como la complejidad lineal asciende casi linealmente mientras que la binaria se encuentra muy por debajo, demostrando la eficiencia de esta técnica.

Finalmente al variar el largo promedio por transacción observamos una curva mucho más pronunciada del método lineal respecto al binario. Existen más items por transacción, pero hay la misma cantidad de transacciones e items, lo que genera un aumento muy significativo en la cantidad de apariciones de un item dentro de nuestra base de datos. Como consecuencia, cada elemento poseerá una mayor cantidad de transacciones que comparar al momento de obtener el soporte relativo, aumentando la complejidad y empeorando la performance de ambos algoritmos.





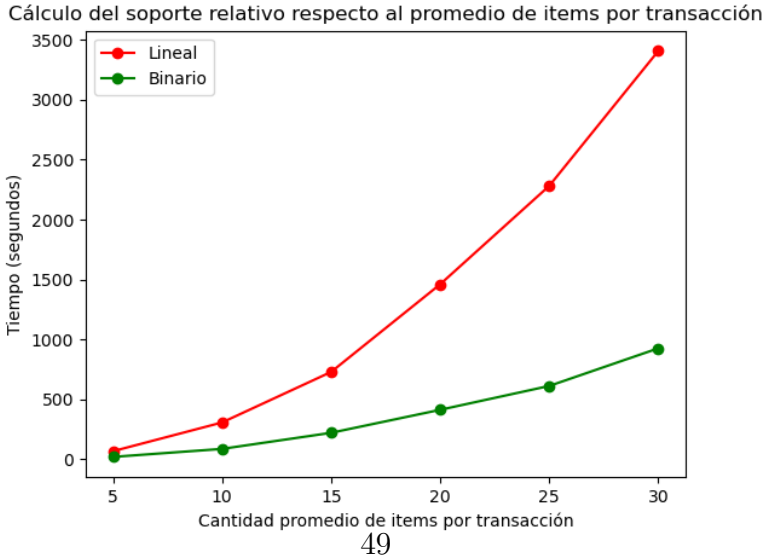


Figura 12: Soporte Relativo

6.4.4. Paralelización en la generación de candidatos

En la sección 4.4.5 hemos distinguido los distintos tipos de algoritmos a evaluar, variando la cantidad de transacciones y la cantidad de items totales se obtuvieron los resultados descriptos en la Figura 13, de la cual podemos hacer varias observaciones.

Paralelizar en los elementos de cada tamaño k es muy ineficiente con respecto a los otros dos algoritmos. Esto es debido a que la cantidad de trabajo a paralelizar no es lo suficientemente compleja para justificar su paralelización. El trabajo en este caso es el cálculo del soporte de cada itemset para un mismo k, que en algoritmos como Cumulate o Apriori implica recorrer la totalidad de las transacciones de cada elemento dentro del itemset para poder concluir con la cantidad total de transacciones en común. Sin embargo en este caso, y como hemos visto en la sección 4.4.4, sólo hace falta recorrer las transacciones que pertenezcan al gránulo temporal donde esté siendo evaluado. Esto reduce significativamente los tamaños del arreglo a comparar y por ende la complejidad de la operación.

Notar que a diferencia del anterior, en el algoritmo de paralelización por nodo (o gránulo) cada hilo de ejecución devolverá todos los itemsets frecuentes de todos los tamaños para un gránulo seleccionado. Este algoritmo posee la mejor performance de los tres, ya que la complejidad del trabajo a realizar justifica su distribución. Sin embargo, esta mejora no es significativa y no se ve afectada al aumentar las transacciones o los items totales.

Si bien los algoritmos sin paralelizar y paralelización por nodo aumentan su tiempo de ejecución a medida que elevamos la cantidad de transacciones, no puede verse una diferencia tan significativa comparada con el algoritmo de paralelización en cada k, ya que este último posee una dificultad base en la paralelización de tareas más simples que es aún mayor al escalar el número de transacciones. Es esperable entonces, que este algoritmo sea más eficiente para un número de transacciones significativamente mayor.

Respecto al número de items totales en nuestra base de datos, podemos ver cómo todos los algoritmos poseen un incremento en su complejidad hasta las 50.000 transacciones, y luego se estabilizan. Esto es producido por la distribución de elementos por transacción. Al existir más items distintos en nuestra base de datos y mantener nuestra cantidad promedio de items por transacción, es menos probable que un ítem aparezca en varias transacciones.

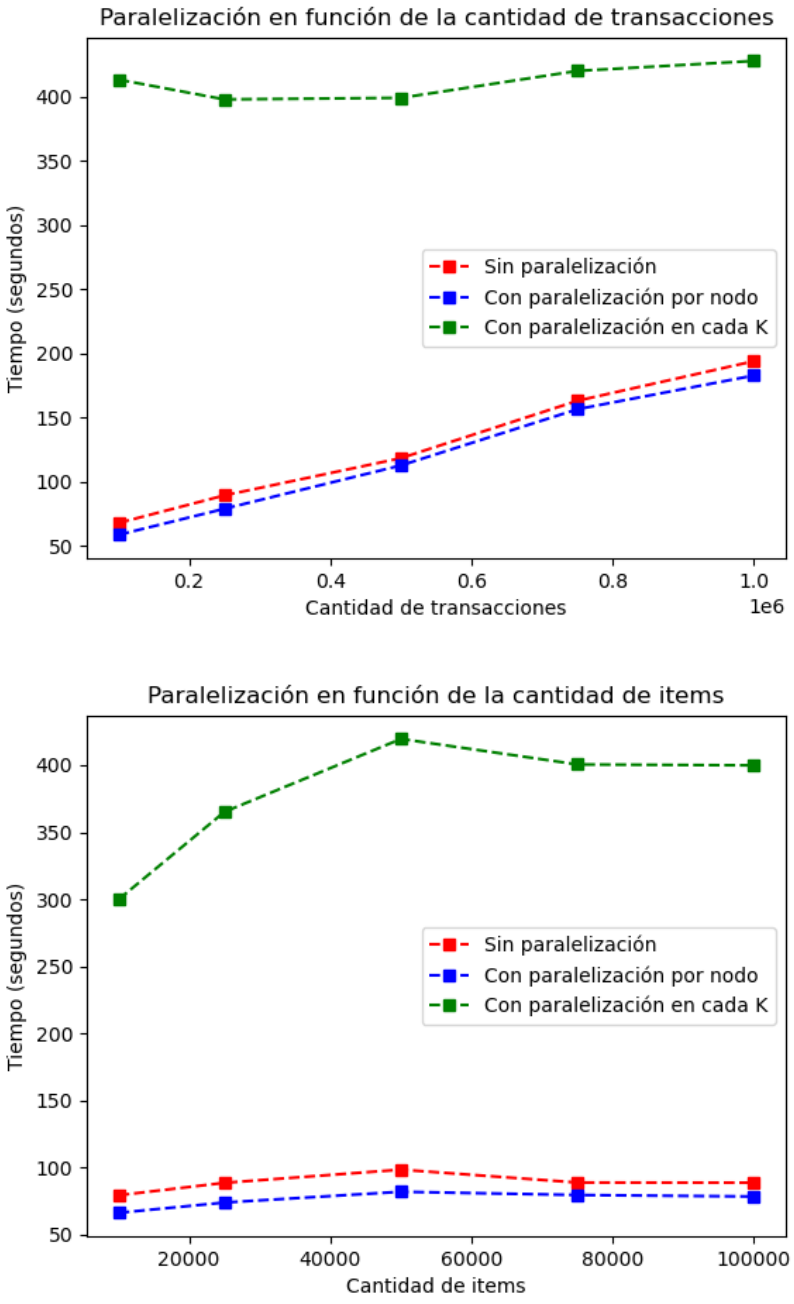


Figura 13: Paralelización en generación de itemsets candidatos

6.4.5. Paralelización en la generación de reglas

Vamos a evaluar los tiempos de la paralelización en la generación de reglas tal cual fue descripto en la sección 4.4.6, variando el soporte mínimo utilizado para generar los itemsets frecuentes que les darán origen. Recordemos que a medida que subamos el soporte mínimo requerido iremos

reduciendo la cantidad de itemsets frecuentes a paralelizar. De los resultados en la Figura 14, se pueden hacer varias observaciones.

Incluso con el soporte más bajo evaluado, la paralelización no es más eficiente que el algoritmo tradicional. Al igual que en el experimento anterior, esto es producido porque la complejidad del trabajo a paralelizar no justifica su distribución con la estrategia elegida. Para generar y evaluar una regla de asociación sólo se requiere dividir los soportes del itemset del lado izquierdo y derecho de la misma. Soportes que ya han sido calculados previamente para cada gránulo y que son accedidos de manera muy rápida al generar las reglas.

El abrupto descenso de complejidad es generado por la volatilidad en la cantidad de itemsets frecuentes a partir de soportes menores a $0,05\,\%$. Incluso con una mayor cantidad de itemsets frecuentes, la diferencia entre ambos no disminuye.

El origen de esta diferencia coincide con la baja performance de la paralelización de reglas obtenida en el algoritmo de *Cumulate*, ya que la razón de fondo sigue coincidiendo. Más detalles de esto en la sección 6.2.1.

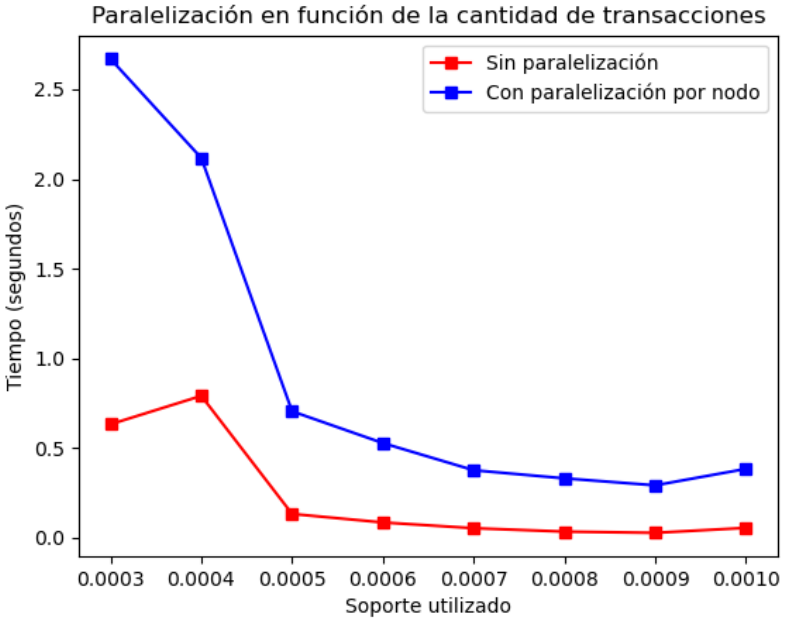


Figura 14: Paralelización en generación de reglas

6.5. Experimentación HTGAR

6.5.1. Comparación *HTGAR* y *Vertical Cumulate*

Observamos la performance del algoritmo *HTGAR* frente a un experimento realizado previamente para *Vertical Cumulate*. Queremos ver que ambos algoritmos se comportan de la misma manera cuando escalamos la taxonomía en cuanto a cantidad de raíces. Como vimos en el experimento **Roots** en la figura [6], aumentar la cantidad de raíces hace que la probabilidad de que una raíz especifica esté en una transacción decrezca. Nuestra hipótesis es que en *HTGAR* ocurre lo mismo sólo que el tiempo total que va a llevar generar los itemsets frecuentes va a ser mayor que en *Vertical Cumulate* ya que tiene que ejecutar la etapa de generación de itemsets frecuentes para cada gránulo temporal. Ejecutamos ambos algoritmos con soporte mínimo en $0,5\,\%$.

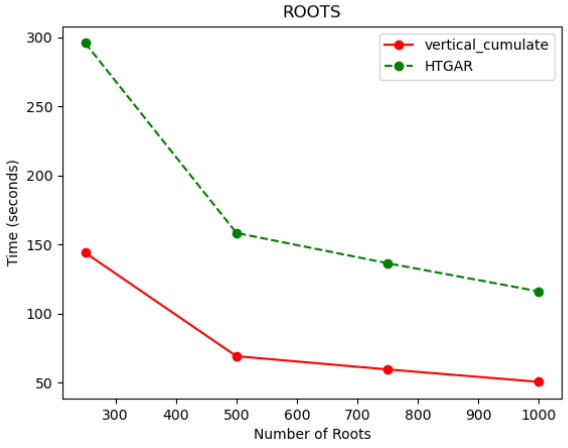


Figura 15: Escalando la taxonomía para HTGAR

6.5.2. Reglas obtenidas por gránulo temporal y filtradas por R interesante

Se realizó un análisis cuantitativo de las reglas generadas con el algoritmo *HTGAR*. La intención es ver cómo se comportan estas reglas con una base de datos real (Foodmart '97) en cada gránulo, con un soporte mínimo de $0,2\,\%$ y una confianza de $75\,\%$. Estos valores fueron elegidos con el fin de obtener un conjunto acotado de reglas. Además vamos a variar el parámetro R interesante para ver como afecta el filtrado de reglas por gránulo temporal.

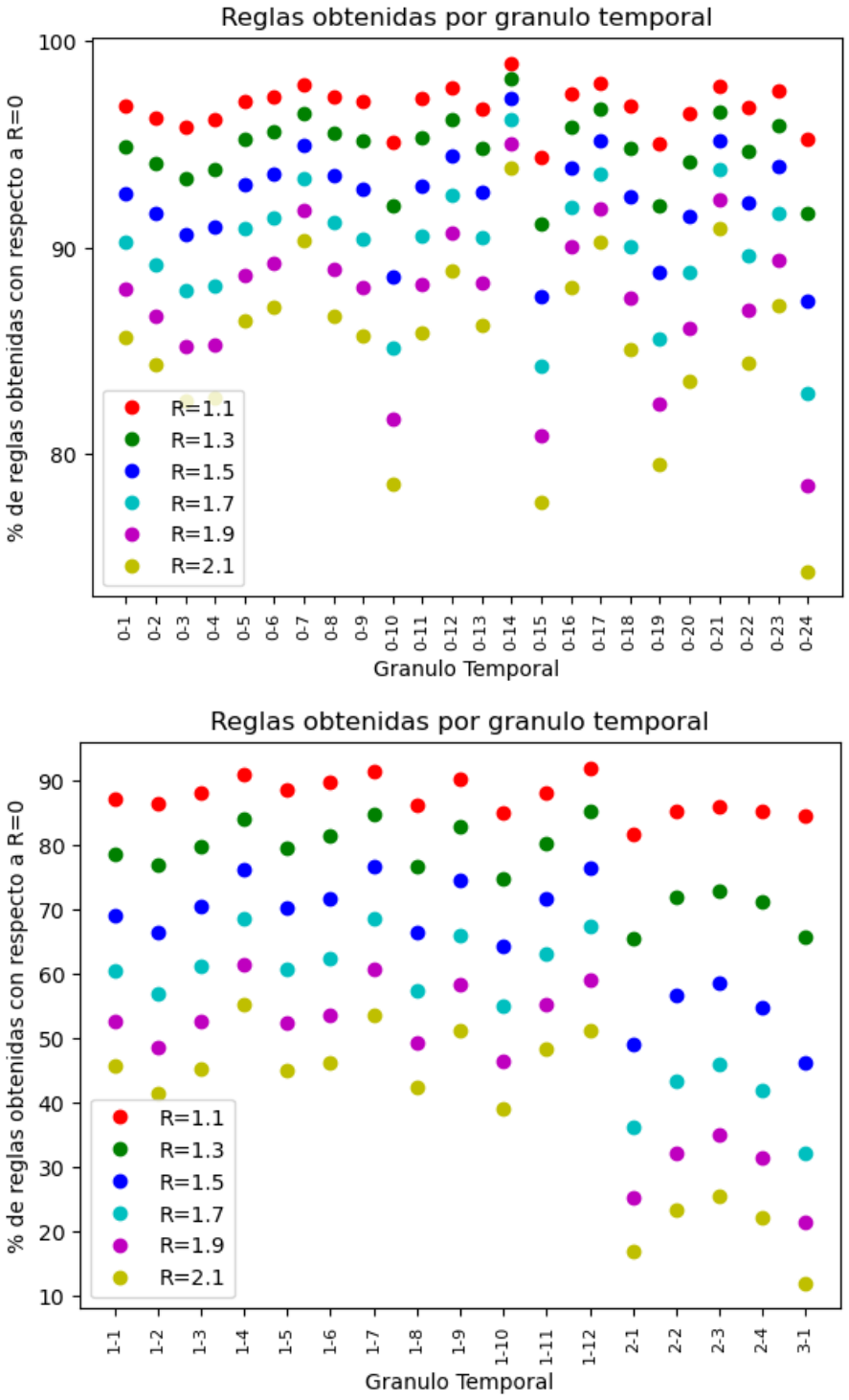


Figura 16: Reglas por gránulo temporal variando el R-interesante por 0 y 1 respectivamente

Con la Figura 16, se pueden hacer varias observaciones

La cantidad de reglas filtradas por el R-interesante aumenta no sólo en la medida que aumenta el parámetro R sino también el nivel jerárquico temporal que se está evaluando. La cantidad de reglas descartadas (o no-interesantes) aumenta significativamente a medida que subimos de nivel hacia la copa del árbol. Así por ejemplo el gránulo $3-1$ conserva el 12 % para R=2.1, y el 80 % de las mismas para cualquier gránulo de nivel 0 (hojas).

Se puede presumir que al aumentar el nivel de jerarquía temporal, se expanden los límites temporales que abarca el período en revisión, involucrando más transacciones en el mismo. En consecuencia, ante un potencial mayor soporte de la regla ancestro (contra la que se compara la regla a descartar) es mucho más probable que la regla evaluada no sea interesante.

La cantidad de reglas aportadas por *Cumulate* (o sea, las que poseen taxonomía) en cada gránulo son significativamente altas. Si bien este incremento acompaña lo observado por los experimentos de *Cumulate* en la sección anterior, como puede observarse en la figura 16 que se encuentra en el Anexo (sección 10) notamos que a medida que se baja de nivel a las hojas, la cantidad de reglas aumenta significativamente. Es por eso por ejemplo que en el gránulo $3-1$ vemos muchas menos reglas que en el gránulo $0-1$.

7. Librería

Todas las implementaciones se almacenaron en un repositorio de Github [3]. La licencia utilizada para el trabajo es del MIT que permite a cualquier persona descargar el repositorio, utilizarlo, modificarlo y distribuirlo sin limitaciones.

Adicionalmente publicamos los algoritmos implementados en Pypi, un repositorio de paquetes para el lenguaje Python. La librería puede descargarse con pip, un instalador de paquetes para Python. Escribimos un manual de uso que se encuentra disponible en el repositorio de Github, en Pypi y en el Anexo 10

[3] https://github.com/Hildorien/TemporalGeneralizedRules.git

8. Conclusiones

Partiendo del artículo original de reglas de asociación y la implementación de *Apriori* [Agrawal et al., 1993] continuamos mejorando el tiempo de ejecución del algoritmo utilizando las técnicas de verticalización de la base de datos y paralelismo. Realizamos la experimentación correspondiente con datos sintéticos que demostraron la efectividad de las modificaciones agregadas. Este análisis nos sirvió para poder aplicar lo aprendido en los otros algoritmos implementados.

Expandimos la familia de reglas de asociación para estudiar las reglas de asociación generalizadas y el algoritmo que las genera llamado *Cumulate* [Srikant and Agrawal, 1995]. Utilizando las mejoras efectuadas en *Apriori* se implementaron dos versiones nuevas a partir de *Cumulate*: *Vertical Cumulate* y *Vertical Cumulate con paralelización* que demostraron una mejora de performance en comparación con el algoritmo original.

Implementamos también una versión de reglas jerárquico-temporales [Hong et al., 2016]. Además de las modificaciones efectuadas al algoritmo *Apriori*, se introdujo la paralelización para intentar obtener una mejora de performance. Los resultados fueron satisfactorios en la paralelización de generación de candidatos en los niveles jerárquicos que no son hoja. Para los hoja sin embargo, la estrategia de paralelización elegida no evidenció resultados satisfactorios

Finalmente, luego de haber estudiado distintas familias de reglas de asociación se decidió combinar las reglas generalizadas con las temporales para generar un nuevo algoritmo que llamamos *HTGAR*. Para estudiar la performance tomamos experimentos realizados previamente con *Cumulate* y *HTAR* para comparar los resultados. Para el caso del experimento tomado de *Cumulate* evidenciamos un comportamiento similar en la performance al escalar un parámetro de la taxonomía. Para el otro caso realizamos un análisis cuantitativo de las reglas halladas utilizando el parámetro *R-interesante* y concluimos que esta nueva medida de interés reduce el conjunto de salida a medida que aumenta el nivel del gránulo temporal.

9. Trabajo futuro

Durante la etapa de experimentación observamos que el conjunto de salida (reglas de asociación) puede ser muy grande, provocando problemas de memoria para el caso de *HTAR* cuando el umbral del soporte es muy bajo. Como trabajo a futuro proponemos agregar parámetros a los algoritmos para reducir el conjunto de salida así como también el formato del mismo.

Las reglas de asociación generalizadas pueden ser utilizadas en otros dominios además de las compras de supermercado como por ejemplo, en el lenguaje natural [Cagliero and Fiori, 2013]. Como trabajo futuro proponemos ampliar el uso de *Cumulate* en otros dominios donde se puedan aplicar técnicas de lenguaje natural para generar las taxonomías de forma automática apoyándose en los resultados hallados en la fase de experimentación de esta tesis.

Existen varias mejoras posibles para el algoritmo de reglas de asociación jerárquico temporal. Se podría trabajar en evaluar distintas particiones temporales dependiendo de la lógica de negocio, así como también jerarquías dinámicas que no están contempladas en el modelo actual.

Otra mejora posible implica optimizar el cálculo de la intersección de las transacciones entre dos itemsets dentro de un mismo gránulo temporal.

A lo largo de este trabajo se indagó en la paralelización como principal herramienta para la mejora de performance. Existen varios factores que alteran la eficiencia de nuestra herramienta. Por mencionar algunos: la memoria utilizada, el compilador, la complejidad de la tarea a paralelizar, el tamaño del objeto iterable, entre otras. Analizar los distintos escenarios que se producen al variar estos factores constituye un nuevo tema de investigación.

Con respecto a la medida R-interesante se puede profundizar el análisis de la relación que existe con la estructura de la taxonomía de los items.

10. Apéndice

R →	0		1.1		1.3		1.5		1.7		1.9		2.1	
Gránulo ↓	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%	Cantidad	%
0-1	294038	100	285125	96,97	279336	95,00	272260	92,59	265441	90,27	258098	87,78	251238	85,44
0-2	203468	100	195899	96,28	191354	94,05	186102	91,46	180927	88,92	175628	86,32	170741	83,92
0-3	239931	100	229835	95,79	223662	93,22	216741	90,33	210053	87,55	203161	84,67	196720	81,99
0-4	249048	100	238219	95,65	230865	92,70	222774	89,45	214441	86,10	206519	82,92	199054	79,93
0-5	294989	100	286216	97,03	280486	95,08	273434	92,69	266467	90,33	259266	87,89	252295	85,53
0-6	365375	100	355349	97,26	348874	95,48	340643	93,23	332314	90,95	323533	88,55	315239	86,28
0-7	556457	100	544148	97,79	535997	96,32	526696	94,65	517142	92,93	507777	91,25	498819	89,64
0-8	197631	100	192295	97,30	188551	95,41	183911	93,06	179330	90,74	174349	88,22	169562	85,80
0-9	411873	100	398753	96,81	389749	94,63	379306	92,09	368304	89,42	357765	86,86	347667	84,41
0-10	125445	100	119426	95,20	115471	92,05	110919	88,42	106446	84,85	101978	81,29	97947	78,08
0-11	297120	100	287672	96,82	281038	94,59	273171	91,94	265120	89,23	257187	86,56	249731	84,05
0-12	452608	100	441888	97,63	434628	96,03	426112	94,15	417082	92,15	408153	90,18	399612	88,29
0-13	335909	100	326934	97,33	321017	95,57	313998	93,48	307186	91,45	299761	89,24	292655	87,12
0-14	936452	100	924411	98,71	915794	97,79	904737	96,61	892213	95,28	878969	93,86	865934	92,47
0-15	191726	100	181768	94,81	175671	91,63	168798	88,04	162308	84,66	155632	81,17	149558	78,01
0-16	293028	100	285723	97,51	280905	95,86	274878	93,81	269111	91,84	262790	89,68	256847	87,65
0-17	500988	100	489678	97,74	482251	96,26	473347	94,48	464229	92,66	454620	90,74	445674	88,96
0-18	187496	100	181500	96,80	177615	94,73	172976	92,26	168129	89,67	163144	87,01	158365	84,46
0-19	147546	100	140117	94,96	135502	91,84	130397	88,38	125557	85,10	120428	81,62	115992	78,61
0-20	222485	100	213954	96,17	208306	93,63	201705	90,66	194960	87,63	188292	84,63	182052	81,83
0-21	524297	100	513649	97,97	506982	96,70	499282	95,23	491670	93,78	483528	92,22	476061	90,80
0-22	354144	100	342422	96,69	334543	94,47	325026	91,78	315343	89,04	305299	86,21	296101	83,61
0-23	389990	100	379933	97,42	372617	95,55	364187	93,38	354293	90,85	344643	88,37	335153	85,94
0-24	160168	100	152046	94,93	146024	91,17	138566	86,51	130850	81,70	123453	77,08	116253	72,58
1-1	42117	100	36936	87,70	33193	78,81	29170	69,26	25537	60,63	22236	52,80	19297	45,82
1-2	48758	100	41839	85,81	36970	75,82	31585	64,78	26869	55,11	22787	46,73	19294	39,57
1-3	57699	100	50907	88,23	45883	79,52	40040	69,39	34484	59,77	29417	50,98	24951	43,24
1-4	65609	100	59532	90,74	54883	83,65	49488	75,43	44415	67,70	39549	60,28	35211	53,67
1-5	56495	100	49800	88,15	44538	78,84	38837	68,74	33526	59,34	28587	50,60	24476	43,32
1-6	57626	100	51273	88,98	46132	80,05	40133	69,64	34576	60,00	29543	51,27	25089	43,54
1-7	78947	100	72431	91,75	67126	85,03	60552	76,70	54024	68,43	47570	60,26	41878	53,05
1-8	45870	100	40005	87,21	35700	77,83	30913	67,39	26654	58,11	22948	50,03	19663	42,87
1-9	56977	100	51119	89,72	46553	81,70	41310	72,50	36474	64,02	31836	55,88	27720	48,65
1-10	38914	100	32893	84,53	28768	73,93	24520	63,01	20781	53,40	17450	44,84	14556	37,41
1-11	66874	100	59346	88,74	53847	80,52	47828	71,52	41896	62,65	36784	55,00	32054	47,93
1-12	63223	100	57955	91,67	53413	84,48	47369	74,92	41358	65,42	35813	56,65	30976	48,99
2-1	31002	100	25380	81,87	20291	65,45	15171	48,94	11214	36,17	7785	25,11	5106	16,47
2-2	42450	100	36044	84,91	30104	70,92	23396	55,11	17783	41,89	13187	31,06	9522	22,43
2-3	37087	100	32079	86,50	27190	73,31	21741	58,62	17158	46,26	13047	35,18	9470	25,53
2-4	36460	100	31118	85,35	25893	71,02	19784	54,26	15234	41,78	11363	31,17	8150	22,35
3-1	30413	100	25699	84,50	20020	65,83	14029	46,13	9763	32,10	6506	21,39	3669	12,06
TOTAL	8788733	100	8461316	96,27	8227742	93,62	7955832	90,52	7690662	87,51	7430381	84,54	7190352	81,81

Figura 17: Heatmap cantidad de reglas obtenidas de experimentación HTGAR

10.1. Manual de librería

Temporal Generalized Association Rules

This library provides four algorithms related to Association Rule mining. You can download this repository as a package with:

pip install TemporalGeneralizedRules

The algorithms are:

- Apriori
- Cumulate
- HTAR
- HTGAR

These algorithms use a transactional dataset that is transformed to a vertical format for optimization.

TGAR

This is the main class that must be instantiated once.

Usage

import TemporalGeneralizedRules

tgar = TemporalGeneralizedRules.TGAR()

Apriori

This algorithm has four parameters: - filepath: Filepath of the dataset in csv format with the format discussed in the previous section.

- *min_supp*. Minimum support threshold
- min_conf. Minimum confidence threshold
- *parallel_count*. Optional parameter that enables parallelization in candidate count phase of the algorithm

Usage

tgar.apriori("dataset.csv", 0.05, 0.5)

Cumulate

This algorithm has six parameters:

- $filepath$. Filepath of the dataset in csv format with the format discussed in the previous section.
- $taxonomy\_filepath$. Filepath of the taxonomy in csv format with the format discussed in the previous section.
- *min_supp*. Minimum support threshold.
- min_conf. Minimum confidence threshold.
- *parallel_count*. Optional parameter that enables parallelization in candidate count phase of the algorithm.

Usage

tgar.cumulate("dataset.csv", 0.05, 0.5, 1.1)

HTAR

This algorithm has four parameters:

- $filepath$. Filepath of the dataset in csv format with the format discussed in the previous section.
- min_supp. Minimum support threshold.
- min_conf. Minimum confidence threshold.
- $parallel\_count$. Optional parameter that enables parallelization in candidate count phase of the algorithm.

Usage

tgar.htar("dataset.csv", 0.05, 0.5)

HTGAR

This algorithm has six parameters:

- $filepath$. Filepath of the dataset in csv format with the format discussed in the previous section.
- $taxonomy\_filepath$. Filepath of the taxonomy in csv format with the format discussed in the previous section.
- min_supp. Minimum support threshold.
- min_conf. Minimum confidence threshold.
- min_r. Minimum R-interesting threshold.
- $parallel\_count$. Optional parameter that enables parallelization in candidate count phase of the algorithm.

Usage

tgar.htgar("dataset.csv", 0.05, 0.5, 1.1)

Pypy

For a better performance we recommend using this package with Pypy, a faster implementation of python.

https://www.pypy.org/

Referencias

[Agrawal et al., 1993] Agrawal, R., Imielinski, T., and Swami, A. (1993). *Mining Association Rules Between Sets of Items in Large Databases, SIGMOD Conference*, volume 22, pages 207–. Agrawal, Rakesh and Imielinski, Tomasz and Swami, Arun.

[Agrawal and Shafer, 1996] Agrawal, R. and Shafer, J. C. (1996). Parallel mining of association rules. *IEEE Trans. on Knowl. and Data Eng.*, 8(6):962–969.

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, page 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Agrawal and Srikant, 1995] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the eleventh international conference on data engineering*, pages 3–14. IEEE.

[Ale and Rossi, 2000] Ale, J. M. and Rossi, G. H. (2000). An approach to discovering temporal association rules. In *Proceedings of the 2000 ACM Symposium on Applied computing-Volume 1*, pages 294–300.

[Cagliero and Fiori, 2013] Cagliero, L. and Fiori, A. (2013). Discovering generalized association rules from twitter. *Intelligent Data Analysis*, 17.

[Chang et al., 2002] Chang, C.-Y., Chen, M.-S., and Lee, C.-H. (2002). Mining general temporal association rules for items with different exhibition periods. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 59–66. IEEE.

[Gupta et al., 2006] Gupta, N., Mangal, N., Tiwari, K., and Mitra, P. (2006). Mining quantitative association rules in protein sequences. In *Mining Quantitative Association Rules in Protein Sequences*, pages 273–281.

[Hong et al., 2016] Hong, T.-P., Lan, G.-C., Su, J.-H., Wu, P.-S., and Wang, S.-L. (2016). Discovery of temporal association rules with hierarchical granular framework. *Applied Computing and Informatics*, 12(2):134–141.

[Li et al., 2003] Li, Y., Ning, P., Wang, X. S., and Jajodia, S. (2003). Discovering calendar-based temporal association rules. *Data & Knowledge Engineering*, 44(2):193–218.

[Ozden et al., 1998] Ozden, B., Ramaswamy, S., and Silberschatz, A. (1998). Cyclic association rules. In *Proceedings 14th International Conference on Data Engineering*, pages 412–421. IEEE.

[Srikant and Agrawal, 1995] Srikant, R. and Agrawal, R. (1995). Mining generalized association rules. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, page 407–419, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Şerban et al., 2006] Şerban, G., Czibula, I. G., and Campan, A. (2006). A programming interface for medical diagnosis prediction. In *A PROGRAMMING INTERFACE FOR MEDICAL DIAGNOSIS PREDICTION*.