Algoritmo de programación genética gramatical para la extracción de reglas de asociación en *Big Data* usando el paradigma MapReduce

Francisco Padillo, José María Luna, Sebastián Ventura, Francisco Herrera

Departamento de informática y análisis numérico, Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España.
{fpadillo,jmluna,sventura}@uco.es,
Departamento de informática e Inteligencia Artificial, Universidad de Granada, 18071 Granada, España.
fherrera@decsai.ugr.es,

Resumen El descubrimiento de reglas de asociación está considerado como una de las tareas más importantes en el análisis y descripción del comportamiento de la información en grandes conjuntos datos. Aunque se han propuesto muchos algoritmos eficientes, el creciente interés en el almacenamiento de datos ha provocado que la cantidad de datos a tratar sea inmanejable. En este sentido, nuestro objetivo es proponer un algoritmo evolutivo basado en gramáticas para la extracción de reglas de asociación sobre MapReduce, haciendo uso de Hadoop y Spark. La experimentación incluye 30 *datasets* y 7 algoritmos diferentes, demostrando el excelente comportamiento de nuestra propuesta sobre diversas métricas de calidad. Se han usado *datasets* de 300 GBytes, demostrando la utilidad de Hadoop y Spark sobre datos de gran tamaño.

Keywords: Reglas de asociación, *Big Data*, MapReduce, Hadoop, Spark

1. Introducción

Con el avance de la tecnología, el almacenamiento de datos está creciendo de forma exponencial, existiendo un aumento de interés en el análisis y extracción de información útil a partir de los datos. Aunque se han propuesto muchas técnicas eficientes para extraer información no trivial a partir de datos en bruto, su rendimiento se está viendo enlentecido debido a la enorme cantidad de datos que deben ser analizados. En la actualidad, se están proponiendo diversas técnicas capaces de trabajar con enormes cantidades de datos de una forma eficiente, englobándose bajo el término *Big Data* [1].

A pesar de que se han propuesto muchas técnicas para el análisis y descripción de los datos [2], la minería de reglas de asociación es una de las más extendidas. Los primeros enfoques tomaron como punto de partida el algoritmo Apriori [3], el cual permite extraer relaciones entre elementos de la base de datos en dos etapas: (1) extracción de patrones frecuentes; (2) extracción de reglas de asociación a partir de los patrones frecuentes previamente obtenidos.

Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial, pp. 137-148

Estas primeras propuestas estaban basadas en el tratamiento de datos discretos, requiriendo el análisis de hasta $2^k - 1$ patrones diferentes cuando se trata con k ítems. Esto hace que, al tratar con datos de alta dimensionalidad, las técnicas basadas en Apriori sean muy ineficientes.

El uso de AEs ha permitido extraer reglas de asociación de manera directa [4], sin necesidad de extraer los patrones frecuentes previamente, y reduciendo el tiempo de cómputo [5]. Además, diversas propuestas basadas en AEs [6,7] han permitido el tratamiento de datos definidos en dominios continuos sin necesidad de un preprocesado previo. Otras propuestas de esta naturaleza han permitido incorporar conocimiento subjetivo y reducir el espacio de búsqueda mediante la aplicación de restricciones sintácticas, entre ellos debemos de destacar G3PARM [8]. Sin embargo, a pesar de que el uso de AEs está muy extendido en el campo de la minería de asociaciones, este tipo de algoritmos pueden no ser eficientes cuando tratamos con *Big Data*.

El objetivo de este trabajo es proponer un algoritmo evolutivo basado en gramáticas para la extracción de reglas de asociación en ámbitos *Big Data* usando MapReduce. Se ha demostrado que el uso de gramáticas en este campo obtiene excelentes resultados tanto en la restricción del espacio de búsqueda como en la posibilidad de introducir conocimiento subjetivo [8]. El algoritmo propuesto, denominado PGG-RA (Programación Genética Gramatical - Reglas de Asociación), ha sido implementado tanto en Hadoop como en Spark, demostrando obtener excelentes resultados en el análisis y tratamiento de *Big Data*. Con el fin de analizar el rendimiento del algoritmo propuesto, se ha realizado un estudio experimental incluyendo 7 algoritmos diferentes y 30 *datasets* de alta dimensionalidad, incluyendo ficheros de más de 300 GBytes. Los resultados obtenidos revelan un excelente comportamiento sobre diversas métricas de calidad.

El resto del artículo está organizado como sigue. Sección 2 presenta algunas definiciones relevantes relacionadas con esta tarea; Sección 3 describe los algoritmos propuestos; Sección 4 presenta los *datasets* usados así como los resultados obtenidos. Finalmente, las conclusiones son expuestas en la Sección 5.

2. Antecedentes

La extracción de reglas de asociación ha recibido especial atención desde su aparición en los años noventa [3]. Su objetivo es el descubrimiento de relaciones útiles y desconocidas entre elementos de la base de datos. De manera formal, una regla de asociación se define como una implicación de la forma $X \rightarrow Y$, donde $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, y $X \cap Y = \emptyset$.

En minería de patrones [9] y, más específicamente, en minería de asociaciones, el interés de las reglas es cuantificado mediante una serie de métricas que permiten determinar el interés de la relación dentro del conjunto de datos. En la literatura existen infinidad de métricas [10], siendo el soporte, la confianza y el *leverage* las más conocidas. El soporte se define como el número de transacciones $\{\forall t_j \in \mathcal{T}\}$ satisfechas por una regla, es decir, $soporte(X \rightarrow Y) = \{\forall t_j \in \mathcal{T} : X \subseteq t_j \wedge t_j \subseteq \mathcal{I}\}$. De forma similar la confianza mide la exacti-

tud de la regla, es decir, $confianza(X \rightarrow Y) = soporte(X \rightarrow Y)/soporte(X)$. A pesar de que estas dos son las métricas más comúnmente usadas en asociación, no permiten detectar la independencia estadística, siendo el *leverage* una de las medidas más útiles en este sentido. El *leverage* se define como $leverage(X \rightarrow Y) = support(X \rightarrow Y) - (support(X) \cdot support(Y))$.

La tarea de extracción de reglas de asociación es una de las más ampliamente utilizadas en el análisis y extracción de conocimiento. Existen numerosas propuestas para tal fin, considerándose Apriori [3] como el primer algoritmo de extracción de reglas, el cual extrae relaciones entre elementos de una base de datos mediante búsqueda exhaustiva. Posteriormente, *Zaki et al.* [11] propuso el algoritmo Eclat, el cual supone una importante mejora sobre Apriori, donde también se sigue un enfoque exhaustivo usando una representación vertical de los datos. A pesar de que muchos de los algoritmos propuestos en la literatura son realmente eficientes, la necesidad de afrontar este complejo espacio de búsqueda en una cantidad de tiempo reducida ha provocado la aplicación de técnicas basadas en AEs [8,6,5]. Sin embargo, a pesar de que los algoritmos existentes en la actualidad han supuesto importantes mejoras, la necesidad de ser aplicadas sobre *Big Data* hace necesaria la adopción de algún tipo de paralelización.

MapReduce [12] es un paradigma reciente para la computación paralela. Éste permite escribir algoritmos paralelos de una forma simple, permitiendo abordar grandes cantidades de datos de una forma eficiente. Los algoritmos en MapReduce están compuestos de dos fases definidas por el programador: *map* y *reduce*; En la fase del *map*, cada *mapper* procesa un subconjunto de los datos de entrada produciendo un conjunto de $\langle k, v\rangle$. Finalmente, el *reducer* obtiene como entrada la salida del *mapper*, produciendo un nuevo conjunto de $\langle k, v\rangle$ finales.

3. PGG-RA: Algoritmo evolutivo basado en gramáticas para la extracción de reglas de asociación en *Big Data*

En esta sección se presenta la propuesta para extraer reglas de asociación en entornos *Big Data*. El mismo enfoque se ha escalado usando plataformas para computación paralela, compartiendo todas ellas la misma idea y obteniendo las mismas soluciones. A pesar de que existan otros algoritmos de Programación Genética Gramatical (PGG) como G3PARM [8], éste no estaba inminentemente diseñado para ser paralelizado como si lo esta PGG-RA. Además, mientras que G3PARM sólo optimiza una única métrica de calidad, nuestra propuesta obtiene excelente resultados optimizando varias métricas.

3.1. Punto de partida

En primer lugar se va a presentar el enfoque general de PGG-RA (Programación Genética Gramatical - Reglas de Asociación), el cual no hace uso de ningún tipo de paralelización. La principal característica de este algoritmo es el uso de gramáticas que permite introducir conocimiento subjetivo.

P = {Regla = Antecedente, Consecuente ;
Antecedente = Condición | Condición, Antecedente ;
Consecuente = Condición | Condición, Consecuente ;
Condición = Numérico | Nominal ;
Numérico = 'nombre' 'EN' 'Min_valor', 'Max_valor' ;
Nominal = 'nombre' '=' 'valor' ; }

Figura 1: Reglas de producción definidas por la gramática de PGG-RA.

PGG-RA representa cada solución como un árbol sintáctico de derivación codificado a través de un conjunto de reglas de producción (Ver Figura 1). Para generar cada solución se aplica una serie de derivaciones, donde el número máximo de derivaciones establece el número máximo de atributos que puede contener una regla. Además, cada una de estas soluciones son evaluadas mediante una función *fitness*. PGG-RA mide el interés de cada regla R mediante el valor de una función F definida como el producto de soporte, confianza y *leverage*, es decir, $F(R) = soporte(R) \cdot confianza(R) \cdot leverage(R)$.

PGG-RA (ver Algoritmo 1), comienza codificando individuos mediante el uso de gramáticas, produciéndose de forma aleatoria un conjunto de soluciones, que son almacenadas en la población principal del algoritmo. Además, nuestro algoritmo incorpora una población auxiliar que permite almacenar las mejores soluciones obtenidas a través del proceso evolutivo. Con el fin de generar nuevos individuos en cada generación del proceso evolutivo, los individuos de la población principal son ordenados por *fitness*. Aquellos individuos con *fitness* similar tendrán mayor probabilidad de ser reproducidos juntos, y se aplican unos operadores de cruce y mutación basados en probabilidad. El operador de cruce

Algoritmo 1 PGG-RA

```
1: P_0 → Inicializar una población aleatoria de N individuos
2: para i = 0 hasta NumeroGeneraciones hacer
3:    Evaluar reglas de P_i
4:    Mantener elitismo a través de la población auxiliar
5:    para cada pareja en OrdenarPorFitness(P_i) hacer
6:       si rand(0,1) < P_cru entonces
7:          pareja ← Aplicar operador de cruce(pareja)
8:       fin si
9:       para cada individuo de la pareja hacer
10:         si rand(0,1) < P_mut entonces
11:            Aplicar operador de mutación(individuo)
12:         fin si
13:      fin para
14:      Guardar nuevos individuos en P_{i+1}
15:   fin para
16:   Añadir población auxiliar a P_{i+1}
17: fin para
```

intercambia subárboles aleatorios entre los dos padres, mientras que el operador de mutación aplica cambios aleatorios en el árbol de derivación. Por último, destacar que PGG-RA también incluye un operador de reparación, encargado de reparar aquellas soluciones no válidas, entendiéndose como soluciones no validas aquellas que no cumplen la restricción $X \cap Y = \emptyset$.

3.2. Escalando PGG-RA en *Big Data* usando computación paralela

Partiendo de PGG-RA como algoritmo base, proponemos el uso de tecnologías emergentes basadas en MapReduce para acelerar el proceso de extracción de reglas de asociación sobre grandes cantidades de datos. Destacar que tanto el enfoque secuencial como los basados en MapReduce obtienen los mismos resultados, siendo la única diferencia el enfoque seguido para su paralelización.

PGG-RA Hadoop. Esta primera versión paralela de PGG-RA para *Big Data* hace uso de MapReduce usando Apache Hadoop. Esta versión hace uso de una estructura MRM (*Map-Redude-Map*) (Ver Figura 2). En la primera fase, cada *mapper* recibe un subconjunto de la base de datos, así como la población principal. Siendo la función de cada uno de los *mappers* la evaluación de la población completa sobre su *sub-dataset*. Como salida de cada *mapper* se produce tantas $\langle k, v\rangle$ como individuos existan en la población principal, donde k representa al individuo y v es una tupla con los valores de soporte de antecedente, soporte de consecuente y soporte de la regla. Una vez que se ejecuta esta fase, el *reducer* es el encargado de recibir estas $\langle k, v\rangle$ y sumar todos los valores de soporte para un mismo individuo. El *reducer* producirá tantas $\langle k, v\rangle$ como individuos existan en la población principal, estas $\langle k, v\rangle$ no son usadas como salida definitiva como se hubiese hecho en un enfoque clásico de MapReduce, sino que se usan como entrada para el segundo tipo de *mapper*, siendo el objetivo de éste triple: (a) los individuos son ordenados en función de su *fitness*, donde aquellos individuos con *fitness* similar tendrán mayor probabilidad de ser reproducidos juntos; (b) mantener el elitismo, actualizando la población auxiliar con los mejores individuos de la población principal, dado que la población principal ya

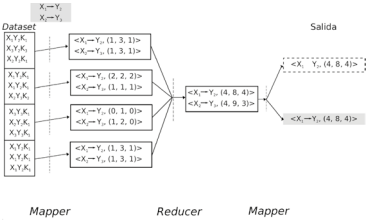


Figura 2: Ejecución de PGG-RA Hadoop sobre un *dataset* de ejemplo, todas las generaciones siguen este mismo procedimiento.

se encuentra ordenada sólo habría que seleccionar los N mejores individuos de esta; (c) generar la nueva descendencia. Como para ordenar la población principal por *fitness* ésta debe de estar almacenada de forma completa sobre memoria principal de un sólo nodo de cómputo, sólo se podría usar un *reducer*. Ya que Hadoop utiliza la salida de cada *reducer* como entrada para el segundo tipo de *mappers*, sólo se podría usar un *mapper* del segundo tipo.

El procedimiento anterior es ejecutado tantas veces como generaciones necesite hacer el algoritmo. De forma que el *dataset* es leído en cada generación desde disco, y la salida de cada generación es escrita en disco. Esto podría provocar un cuello de botella, siendo más eficiente realizar una sola lectura desde disco y almacenar el *dataset* en memoria principal para acelerar futuras lecturas.

PGG-RA Spark. Esta versión de PGG-RA hace uso de MapReduce a través de Apache Spark. El enfoque seguido por este algoritmo es similar al usado por Apache Hadoop, siendo la principal diferencia que en esta versión sólo se lee el *dataset* una sola vez desde disco. Además, la salida de cada una de las generaciones no es almacenada en disco como ocurría en el caso anterior sino en memoria. Tal y como muestra la Figura 3, este algoritmo comienza leyendo el *dataset* desde disco, cargando éste en una estructura de datos especial (RDD) que permite almacenar archivos de forma distribuida dividiéndolos en partes sucesivas. Tras esto, se ejecuta un *mapper* para cada una de las particiones, siendo su función evaluar los individuos de la población principal (cuadro sombreado) para cada uno de sus *sub-dataset*, produciendo cada *mapper* tantas $\langle k, v\rangle$ como individuos existan en la población principal. Donde k es usado para representar al individuo, y v representa a una tupla con los valores de soporte de antecedente, soporte de consecuente y soporte. Posteriormente, el *reducer* calculará los valores totales para cada uno de los soportes. Por último, el *driver* recibe la población principal evaluada, siendo su función la de realizar la selección, reproducción y creación de la nueva población. Una vez que la nueva población este creada, ésta será distribuida para su evaluación, repitiéndose el proceso anterior. A diferencia de Hadoop, esta versión no necesita leer el *dataset* desde disco porque ya se encuentra en memoria principal (RDD).

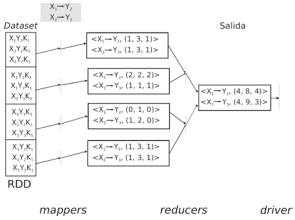


Figura 3: Ejecución de PGG-RA Spark sobre un *dataset* de ejemplo para la primera iteración, esta iteración es la única que lee el dataset desde disco.

Dataset	Atributos(R/I/N)	Instancias
Datasets reales		
Tic-Tac-Toe	9 (0/0/9)	958
Flare	11(0/0/11)	1,066
Car	6 (0/0/6)	1,728
Chess	36 (0/0/36)	3,196
Ring	20 (20/0/0)	7,400
Twonorm	20 (20/0/0)	7,400
Mushroom	22 (0/0/22)	8,124
Coil2000	85 (0/85/0)	9,822
PenBased	16 (0/16/0)	10,992
Nursery	8 (0/0/8)	12,690
Magic	10 (10/0/0)	19,020
Letter	16 (0/16/0)	20,000
UJIIndoorLoc	529 (2/527/0)	21,048
House16H	17 (10/7/0)	22,784
Grammatical	100 (100/0/0)	27,965
ChessKrKp	6 (0/0/6)	28,056
Adult	14 (6/0/8)	48,842
Statlog (Shuttle)	10 (0/10/0)	58,000
Connect4	42 (0/0/42)	67,557
ColorTexture	17 (16/1/0)	68,040
ColorHistogram	33 (32/1/0)	68,040
Fars	29 (5/0/24)	100,968
Census	41 (1/12/28)	299,284
Covtype	54 (0/10/44)	581,012
Transactions90k	3 (0/3/0)	855,367
Poker	10 (0/10/0)	1,025,010
US Census Data 1990	68 (0/0/68)	2,458,285
SUSY	18 (18/0/0)	5,000,000
HEPMASS	28 (28/0/0)	10,500,000
HIGGS	28 (28/0/0)	11,000,000
Datasets sintéticos		
Sintético	8 (4/0/4)	10^9
Sintético	16 (8/0/8)	10^9
Sintético	24 (12/0/12)	10^9
Sintético	32 (16/0/16)	10^9
Sintético	40 (20/0/20)	10^9
Sintéticos	48 (24/0/24)	10^5 hasta 10^9

Tabla 1: *Datasets* usado para la fase de experimentación

4. Experimentación

El objetivo de esta sección es analizar tanto la eficacia como la eficiencia de nuestra propuesta, analizando tanto la versión secuencial, como la de Hadoop y Spark. El objetivo de esta sección es doble. (1) Demostrar la convergencia de

nuestro algoritmo comparándolo con algoritmos basados en *itemsets*. Adicionalmente, se han incluido comparaciones con 7 algoritmos ampliamente conocidos, los cuales pueden dividirse en dos categorías: basados en *itemsets* (Apriori [3] y Eclat [11]) y AEs (GAR [13], GENAR [6], EARMGA [7], G3PARM [8], QAR-CIP-NSGA-II [5]). (2) Analizar la escalabilidad de nuestra propuesta usando *Big Data*.

El análisis de los algoritmos se ha llevado a cabo sobre 30 *datasets*, cuyo número de atributos varía entre 6 y 529; y el número de instancias varía desde 5 620 hasta 11 000 000. Además, se han incluido ejecuciones con *datasets* sintéticos, donde el número de instancias varía desde $1 \cdot 10^5$ hasta $1 \cdot 10^9$ (incrementando un orden de magnitud) y con un número de atributos desde 8 hasta 48, con un tamaño de archivo de hasta 296 GBytes. La Tabla 1 muestra las características de estos *datasets*. Es importante destacar que los *dataset* reales se han usado para comparar la eficacia de nuestra propuesta con otros algoritmos, mientras que los *datasets* sintéticos se han usado para comprobar la escalabilidad.

Tras realizar un estudio experimental se ha determinado que el número de generaciones para PGG-RA es de 1000, el proceso evolutivo será detenido si los individuos de la elite no mejoran durante 75 generaciones seguidas. Usando 200 individuos para la población principal y 20 para la población auxiliar. Los operadores de reproducción son aplicados con una probabilidad de 0.8.

4.1. Comparativa con algoritmos basados en *itemsets*

El objetivo de este estudio es demostrar que PGG-RA obtiene las mejores soluciones en un tiempo razonable. En este sentido, se ha realizado una comparativa con algoritmos de búsqueda exhaustiva, Apriori y Eclat. Ambos obtienen los mismos resultados, siendo la única diferencia el enfoque seguido. Para realizar la comparativa, se han obtenido todas las reglas y se han ordenado de acuerdo a la función $F(R) = soporte(R) \cdot confianza(R) \cdot leverage(R)$, seleccionándose las 20 mejores. Este número es el mismo que PGG-RA devuelve para cada ejecución.

En este estudio se han incluido 10 *dataset* reales, donde PGG-RA obtuvo el mejor individuo posible en todos ellos. Mientras que cuando se considera la media de las 20 soluciones, PGG-RA obtuvo la misma media de F que los algoritmos de búsqueda exhaustiva para 7 de ellos. Para comprobar si existen diferencias significativas entre los algoritmos de búsqueda exhaustiva y nuestra propuesta se ha usado un Test de Wilcoxon. Obteniéndose un *p-value* de 0.19, luego se puede afirmar que no existen diferencias significativas entre un algoritmo de búsqueda exhaustiva y PGG-RA en cuánto a eficacia. Pero lo mejor de todo, es que PGG-RA consigue obtener los mismos resultados que un algoritmo de búsqueda exhaustiva pero en un menor tiempo. En el peor de los casos, los algoritmos de búsqueda exhaustiva obtienen las mismas soluciones que PGG-RA usando un 1 % del tiempo de este último, mientras que en el mejor de los casos necesitan un 1 300 000 % del tiempo que obtiene PGG-RA.

4.2. Comparativa de PGG-RA con otros AEs

Una vez que se ha demostrado que PGG-RA converge al óptimo global, se va a comparar con otros AEs. Se han realizado 10 ejecuciones para cada *dataset* y los AEs se han ejecutado sin realizar ningún tipo cambio, usándose la versión original propuesta por los autores. La Tabla 2 muestra los rankings medios obtenidos, donde cada columna representa un algoritmo y cada fila representa una métrica diferente. Tal y como se puede apreciar, PGG-RA obtiene los mejores resultados para *Leverage*, *YulesQ* y *NetConf*. Además, aunque el ranking medio en Confianza pueda no parecer tan alto, todos los valores de Confianza obtenida por nuestra propuesta son mayores de 0.9, incluso en algunos casos específicos están por encima de 0.98. Por tanto, es posible afirmar que el comportamiento de PGG-RA para esta métrica es bueno a pesar del valor de ranking obtenido. Cuando se considera el ranking para la métrica del Soporte se obtiene un valor medio con respecto al resto de algoritmos, es importante destacar que valores altos de soporte producen reglas obvias o no interesantes, además, valores muy altos de esta métrica puede producir valores extremadamente malos en el resto de métricas. Luego, como es necesario un compromiso entre el valor de soporte y las otras métricas se podría afirmar que PGG-RA se comporta realmente bien.

Con el objetivo de comprobar si todos los algoritmos se comportan de igual forma para cada una de las métricas se ha realizado un test estadístico de Friedman. Donde $F_{0.01,29,145} = 3.1459$, luego es posible rechazar la hipótesis nula de que todos los algoritmos se comportan de igual forma desde que los valores de F_F son: *Leverage* 53.36, Confianza 22.09, Soporte 23.63, *Lift* 28.42, *YulesQ* 22.10, *NetConf* 43.56. Para comprobar entre que algoritmos existen estas diferencias significativas se ha usado un test estadístico de Bonferroni-Dunn, con

Métrica	PGG-RA	GAR	GENAR	EARMGA	G3PARM	QAR_CIP_NSGA_II
Leverage	**1.00**	3.62	4.57	4.92	4.75	2.87
Confianza	4.51	4.73	3.48	**1.61**	2.11	4.53
Soporte	3.01	2.73	4.83	3.98	**1.55**	4.88
Lift	2.2	4.1	4.18	4.26	4.88	**1.38**
YulesQ	**1.67**	3.48	4.13	5.09	4.26	2.34
NetConf	**1.6**	3.40	4.15	5.18	4.82	1.86

Tabla 2: Ranking medios tras realizar 10 ejecuciones sobre 30 *datasets* reales. Valores en negrita muestra el mejor algoritmo

PGG-RA vs	Victorias	Derrotas	Empates
GAR	4	0	2
GENAR	5	0	1
EARMGA	4	1	1
G3PARM	4	1	1
QAR_CIP_NSGA_II	2	0	4

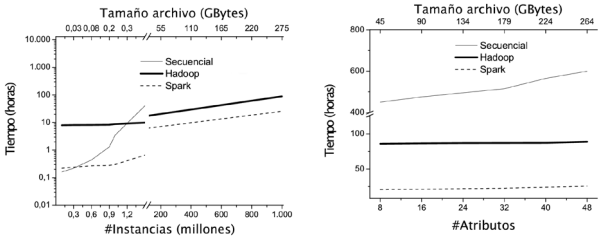
Tabla 3: Diferencias estadisticas de acuerdo al Test de Bonferroni-Dunn.

un $\alpha = 0.01$ y una diferencia crítica de 1.82. Los resultados de este análisis se pueden encontrar en la Tabla 3, donde se puede apreciar que PGG-RA supone una considerable mejora con respecto a GAR, GENAR, EARMGA y G3PARM. Obteniendo un rendimiento más igualatorio con respecto a QAR_CIP_NSGA_II, sin embargo, mejora los resultados de este algoritmo para dos de las métricas.

4.3. PGG-RA en *Big Data* con Apache Hadoop y Spark

El objetivo de esta sección es estudiar la escalabilidad de nuestra propuesta usando diferentes implementaciones sobre datos de diferentes dimensionalidades. En este sentido, la Figura 4a muestra que la implementación secuencial es la más eficiente cuando el tamaño de los datos es pequeño (por debajo de $4 \cdot 10^5$). Sin embargo, cuando el número de instancias crece, su rendimiento se ve empeorado. En este punto, enfoques paralelos comienzan a ser interesantes. PGG-RA Spark se convierte en el más eficiente. Cuando el número de instancias se vuelve a incrementar desde 10^6 hasta 10^8, el rendimiento del enfoque secuencial carece de sentido teniendo un crecimiento exponencial. Sin embargo, las implementaciones basadas en arquitecturas de *Big Data* se comportan realmente bien aunque el tamaño de archivo sea de casi 300 GBytes. En concreto, Apache Spark es la implementación más eficiente suponiendo una considerable mejora con respecto a Apache Hadoop desde que no necesita escribir los resultados, ni tampoco leer el *dataset* desde disco en cada una de las generaciones. Es posible que Spark pudiese comportase peor o igual que Hadoop, cuando este no tenga suficiente memoria, ya que en este caso tendría que realizar la misma dinámica que Hadoop.

Continuando este estudio con un enfoque diferente, se va a analizar como afecta el número de atributos en el rendimiento. La Figura 4b muestra el rendimiento de los algoritmos cuando el número de atributos cambia, tal y como se puede apreciar el número de atributos no parece afectar en gran medida a Spark o Hadoop siendo su rendimiento casi constante conforme aumenta el número de



(a) Rendimiento cuando el número de instancias varía desde 10^5 hasta 10^9.

(b) Rendimiento cuando el número de atributos varia desde 8 hasta 48.

Figura 4: Rendimiento sobre *datasets* sintéticos

atributos. En el caso del secuencial su rendimiento se ve más empeorado que las otras implementaciones ya que un número mayor de atributos implica un *dataset* mayor, estando esta implementación más afectada por el tamaño de archivos.

5. Conclusión

En este trabajo se ha propuesto el algoritmo PGG-RA, un algoritmo evolutivo de programación genética gramatical para la extracción de reglas de asociación sobre *Big Data*. El uso de gramáticas permite al experto introducir conocimiento subjetivo en el proceso de búsqueda, así como reducir el espacio de búsqueda.

Además, se han propuesto dos implementaciones para *Big Data* basadas en MapReduce, usando tanto Apache Hadoop como Apache Spark. Ambas implementaciones devuelven el mismo conjunto de reglas.

El estudio experimental realizado incluye 7 algoritmos y 30 *datasets* de alta dimensionalidad, incluyendo archivos de más de 300 GBytes. Los resultados obtenidos revelan un excelente comportamiento sobre diversas métricas de calidad, demostrando que PGG-RA obtiene un conjunto reducido de reglas con pocos atributos en una cantidad de tiempo razonable sobre *big data*.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Innovación y Ciencia, y los fondos FEDER, bajo el proyecto TIN-2014-55252-P.

Referencias

1. S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. Benítez, and F. Herrera, "Data discretization: Taxonomy and big data challenge," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2015.
2. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2000.
3. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.
4. S. Ventura and J. M. Luna, *Pattern Mining with Evolutionary Algorithms*. Springer International Publishing, 2016.
5. D. Martín, A. Rosete, J. Alcalá-Fdez, and F. Herrera, "Qar-cip-nsga-ii: A new multi-objective evolutionary algorithm to mine quantitative association rules," *Information Sciences*, vol. 258, pp. 1–28, 2014.
6. J. Mata, J. L. Alvarez, and J. C. Riquelme, "Mining numeric association rules with genetic algorithms," in *Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms*, ser. ICANNGA 2001, Taiwan.
7. X. Yan, C. Zhang, and S. Zhang, "Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support," *Expert Systems with Appications*, vol. 36, pp. 3066 – 3076, 2009.
8. J. M. Luna, J. R. Romero, and S. Ventura, "Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules," *Knowledge and Information Systems*, vol. 32, no. 1, pp. 53–76, 2012.

9. J. M. Luna, "Pattern mining: current status and emerging topics," *Progress in Artificial Intelligence*, pp. 1–6, 2016.
10. P. Tan and V. Kumar, "Interestingness Measures for Association Patterns: A Perspective," in *Proceedings of the Workshop on Postprocessing in Machine Learning and Data Mining*, ser. KDD '00, New York, USA, 2000.
11. M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.
12. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM - 50th anniversary issue*, 2008.
13. J. Mata, J. L. Alvarez, and J. C. Riquelme, "Discovering numeric association rules via evolutionary algorithm," in *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, ser. PAKDD 2002, Taiwan.