Detección de outliers en grandes bases de datos mediante aproximación basada en celdas

Adrián De Armas[1], Mag. Bibiana D. Rossi[1], Dr. Horacio Kuna[2]
[1] Universidad Argentina de la Empresa, Facultad de Ingeniería y Ciencias Exactas, Buenos Aires, Argentina.
adearmas@uade.edu.ar, birossi@uade.edu.ar
[2] Universidad Nacional de Misiones, Facultad de Ciencias Exactas Químicas y Naturales. Misiones. Argetina.
hdkuna@unam.edu.ar

Resumen: Este artículo aborda la problemática de la detección de outliers en grandes bases de datos. En base a la aproximación por celdas propuesta por Edwin Knorr y Raymond NG en 1998 en el trabajo "Algorithms for Mining Distance-Based Outliers in Large Datasets" se implementaron distintas versiones del algoritmo que superan las limitaciones establecidas en el trabajo original con modificaciones orientadas a mejorar la eficiencia y la utilización del algoritmo en distintos escenarios.
Palabras clave: Outliers, celdas, algoritmos basados en distancia, grandes bases de datos, paralelización, determinación probabilística, FindAllOutsM.

1. Introducción

Un outlier es cualquier dato que parece estar fuera de lugar respecto al resto de los datos. En la bibliografía pueden encontrarse sinónimos tales como excepción, contaminante, disidente, desvío, anomalía, valor discordante, sorprendente o inesperado[1]. Una de las definiciones más citadas es: "Un outlier es una observación que se desvía tanto de otras observaciones que despierta la sospecha de haber sido generado por un mecanismo diferente" [2].

Para algunas aplicaciones, los eventos raros son interesantes. Aplicaciones como la detección de fraude con tarjeta de crédito y el monitoreo de actividades criminales en el comercio electrónico se benefician con la detección de outliers. Por ejemplo, en el comercio electrónico se espera la existencia de muchas transacciones de bajo valor, sin embargo son los casos excepcionales (importe de la transacción, tipo de compra, hora en que se realiza, ubicación o alguna combinación de las anteriores) lo que interesa tanto para la detección de fraude o por motivos de marketing.

Distintos métodos de detección de outliers han sido propuestos. Métodos de análisis de valores extremos, modelos estadísticos y probabilísticos, métodos basados en profundidad, desvío, ángulos o distancia son algunos de ellos. En este artículo se presentan las mejoras significativas realizadas al algoritmo **FindAllOutsM** de aproximación por celdas propuesto por Edwin Knorr y Raymond NG en 1998 [3].

El artículo se organiza de la siguiente manera: la sección 2 expone la problemática para la detección de outlieres y presenta la estrategia basada en celdas. En la sección 3

se presentan las mejoras propuestas detallando las distintas implementaciones del algoritmo logradas. Finalmente, la sección 4 se reseñan las conclusiones del trabajo.

2. Problemática actual

El tamaño del conjunto de datos y el número de dimensiones han comprobado ser obstáculos claves para el análisis de los datos. La minería de datos provee algoritmos que son escalables tanto en el tamaño del conjunto de datos como en las dimensiones (variables) a evaluar [4].

La mayoría de los trabajos existentes sobre la detección de outliers yacen en el campo de la estadística. Consistentemente, una centena de formas de detectar outliers han sido desarrolladas para diferentes circunstancias, dependiendo de [5] [6]:

- La distribución de los datos
- Si los parámetros de la distribución son conocidos o no
- El número de outliers esperados
- El tipo de outliers esperados

Sin embargo, estas formas de detección presentan dos inconvenientes:

1. Casi todas son univariadas. Esta restricción hace que sean inaplicables para grupos de datos multidimensionales.
2. Todas son basadas en la distribución de los datos. Existen numerosas situaciones donde se desconoce si un atributo en particular sigue una distribución normal, una distribución gama, o cualquier otra, por lo cual, se deben realizar pruebas extensas para encontrar una distribución que se ajuste al atributo.

En el año 1998 Edwing Knorr y Raymond NG publicaron el trabajo "*Algorithms for Mining Distance-Based Outliers in Large Datasets*. Lo interesante de la propuesta de Knorr y Raymond fue que la detección de outliers basados en la distancia de un objeto a sus vecinos más cercanos, es no paramétrica, es decir, no se basa en alguna distribución específica [3][7] y se propone su uso para bancos de datos multivariados de 4, a lo sumo 5 atributos.

Distintos métodos se han propuesto y discutido, a partir de entonces, para la detección paramétrica y no paramétrica de outliers como se puede leer en los trabajos referenciados como [8][9][10][11][12].

2.1. Detección de outliers basada en celdas.

La detección de outliers basada en celdas, utiliza el método de determinación de outliers por distancia. En este artículo se abordan los métodos basados en distancia (Distance Based o *DB*) bajo la noción que un outlier se define como:

"Un objeto O en un grupo de datos T es un oulier $DB(p, D)$ si al menos una fracción p de los objetos en T se encuentran a una distancia mayor que la distancia D (umbral) respecto de O" [1].

Por ejemplo, si el banco de datos en el que se buscan outliers contienen 1000 objetos, con $p = 0.975$ y $D = 10$, se calcula el umbral de outliers, esto se realiza mediante la

siguiente fórmula: $M = N(1-p) = 1000\ (1-0{,}975) = 1000 * 0{,}025 = 25$, luego, todo objeto del banco de datos es un outlier si la cantidad de objetos a una distancia D del objeto a analizar es menor o igual a 25.

El método basado en celdas es exponencial en dimensionalidad pero lineal en puntos de datos. En esta técnica el espacio de datos se divide en celdas, el ancho es una función del umbral D y la dimensionalidad de los datos. Cada dimensión se divide en celdas de ancho $\frac{D}{(2\sqrt{d})}$. La presencia de puntos en una celda así, como en celdas adyacentes, satisface ciertas propiedades que son explotadas para una mayor eficiencia de procesamiento [14]. La aproximación puede explicarse más fácilmente para 2 dimensiones. Se forma una grilla con celdas sucesivas que se encuentran a una distancia máxima de $\frac{D}{(2\sqrt{d})}$. Algo importante a tener en cuenta es que el número de celdas se basa en la partición del espacio de datos y es independiente del número de datos.

Para una celda dada, los vecinos L_1 están definidos por el conjunto de celdas que son alcanzables por esa misma celda cruzando tan solo una celda. Los vecinos L_2 son aquellas celdas que se obtienen al cruzar 2 o 3 celdas. Un caso particular de una celda marcada con X junto con sus vecinos L_1 y L_2 puede observarse en la figura 1 [13].

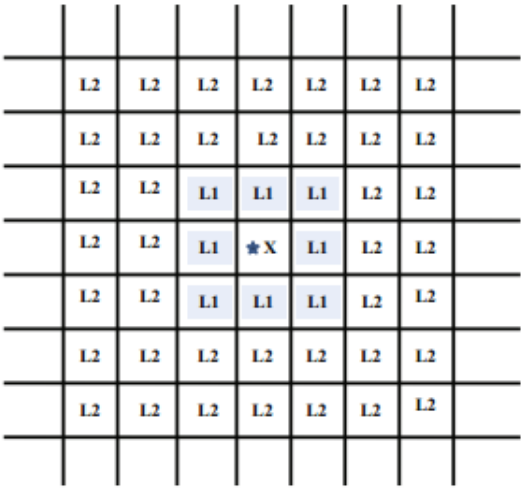

Figura 1 – Particiones del espacio de datos basado en celdas [13]

La celda marcada con una X tiene 8 vecinos L_1 y 40 vecinos L_2. Teniendo esto en cuenta las siguientes propiedades se pueden observar [1]:

- La distancia entre un par de puntos en una celda es como máximo $D/2$
- La distancia entre un punto y un punto vecino L_1 es como máximo D
- La distancia entre un punto y un punto vecino L_r $(con\ r > 2)$ es al menos D

Las únicas celdas por las cuales no se pueden sacar conclusiones inmediatas son aquellas en L_2. La zona L_2 representa una región de incertidumbre para cuyos objetos el cálculo de la distancia necesita ser realizado de forma explícita [13].

Al mismo tiempo un número de reglas pueden ser definidas con el fin de inmediatamente declarar algunas fracciones de los datos como outliers o no outliers. Siendo M la cantidad umbral de objetos para la determinación de outliers, las reglas son [1]:

- Si más de M puntos están contenidos en una celda A, ninguno de los objetos en A es un outlier.
- Si más de M puntos están contenidos en una celda $A \cup L_1(A)$, ninguno de los objetos en A es un outlier
- Si menos o exactamente M puntos están contenidos en una celda $A \cup L_1(A) \cup L_2(A)$, entonces *todos* los puntos en la celda A son outliers

Las celdas se dividen en tres colores, rojas, rosadas y blancas. Las celdas rojas son aquellas cuya cantidad de objetos contenidos en si misma supera el umbral M. Las celdas rosas son aquellas celdas que se encuentran en el vecindario L_1 de una celda roja y cuya cantidad de objetos contenidos es menor al umbral M. Las celdas blancas son aquellas que no son ni rojas ni rosadas. El rendimiento del algoritmo mejora a medida que hay mayor proliferación de celdas rojas o rosadas ya que los objetos de éstas son descartados como outliers. Para todos los objetos en celdas blancas la determinación de su condición de outliers se realiza calculando las distancias del objeto a todos sus vecinos incluidos en la misma celda más los vecindarios L_1 y L_2 . El método se explica detalladamente en el trabajo "Outliers and data mining" [1].

Mediante este método cada elemento del banco de datos adquiere la etiqueta de outlier o no. La falta de ranking de outliers, es una debilidad conocida del método, en el trabajo "*LOF: Identifying Density-Based local Outliers"* [9] se muestra, la debilidad de los métodos basados en distancia al identificar ciertos *tipos* de outliers.

3. Solución

El algoritmo **FindAllOutsM** se implementó utilizando Embarcadero RAD Studio XE. El lenguaje utilizado para escribir los programas fue Object Pascal. En el trabajo original se propone el uso de esta técnica para bancos de datos multidimensionales de 4 dimensiones (cantidad de atributos evaluados), o a lo sumo 5. Se comprobaron las limitaciones del algoritmo[14], ejecutándolo con NHL94, y utilizando un banco de datos de estadísticas de la Liga Nacional de Hockey del año 1994 (uno de los conjuntos de datos utilizados en el trabajo original [3]), junto con los correspodientes valores de p y D. Una vez confirmadas las limitaciones del algoritmo, se introdujeron mejoras a las distintas implementaciones realizadas para extender la cantidad de escenarios en los que se puede usar **FindAllOutsM.** Las mejoras introducidas son:

1. Virtualización del hipercubo propuesto por el algoritmo
2. Persistencia de la relación objeto/celda en un almacenamiento secundario
3. Paralelización del algoritmo
4. Determinación probabilística de outliers

Durante la realización de los experimentos [14] se consideró más eficiente un programa que arroja resultados correctos más rápido. Si dos implementaciones del mismo algoritmo arrojan resultados en tiempos parecidos (+/- 5%), es considerado más eficiente aquel que menos memoria RAM requiera para brindar los resultados.

Para poder medir el rendimiento del algoritmo se utilizó un único banco de datos para todas versiones desarrolladas. Sólo se modificaron los parámetros de p y D para obtener resultados relevantes. El algoritmo debe ser capaz de brindar resultados basándose en la cantidad de atributos a procesar, el valor de p y el valor de D independientemente de la cantidad de filas que conformen el banco de datos. Cada vez que se introducía una mejora a la implementación del algoritmo, se comprobaba el correcto funcionamiento de la misma mediante la comparación de los resultados arrojados contra outliers detectados por fuerza bruta. Una vez confirmado el correcto funcionamiento de la nueva versión, se documentaba su desempeño para distinta cantidad de atributos y tamaño del banco de datos.

El motor de base de datos utilizado para el experimento es FirebirdSQL. Firebird es una base de datos relacional que ofrece muchas de las características del estándar ANSI SQL y puede ejecutarse en Linux, Windows y una variedad de plataformas Unix. Algunas de las mejoras propuestas al programa que implementa el algoritmo **FindAllOutsM** requieren de la utilización de archivos indexados. Esta característica fue implementada mediante el uso de la base de datos SQLite versión 3.

La base de datos Firebird fue instalada en una notebook Dell Studio 1555 (Intel Core 2 Duo P8600 2.4 GHz / ATI Mobility Radeon HD 4570) con 3 Gb de memoria, con 500Gb de disco y con Ubuntu 12.04 como sistema operativo.

La PC de prueba fue una notebook HP Intel® Core™ i7-4700MQ CPU@2.4GHz con 16GB de memoria, 1TB de disco y sistema operativo Windows 8.1 Pro.

3.1. Virtualización del hipercubo de celdas.

Para ejecutar el algoritmo **FindAllOutsM** con 6 atributos, en un banco de datos de 871 objetos, la implementación original requiere más de 60 millones de celdas (la cantidad de celdas depende del banco de datos utilizado) que se mantienen en memoria. Durante la experimentación [14] se comprobó que, para sistemas de 32 bits, no se puede reservar memoria suficiente para las celdas, la instancia de cada una y los objetos que se están procesando. La mejora al algoritmo implica, en este caso, sólo crear las celdas necesarias (sin reservar memoria previamente), para aquellas que tengan objetos asociados o deban ser marcadas con un color diferente al blanco. Lo que permite identificar cada celda es su posición relativa dentro del hipercubo, por ejemplo $[0,4,0]$. Para lograrlo, se implementó una lista de doble entrada: por un lado una cadena que representa las coordenadas de la celda y por el otro la posición (índice) que ocupa la celda dentro de una lista de celdas (objetos). Para identificar la ubicación de una celda mediante sus coordenadas se utilizó un diccionario definido como un "hashtable". Los hashtables representan una colección de pares de datos definidos como clave (coordenadas de la celda) y valor (posición de la celda en la lista de celdas), que se organizan en base al código hash de la clave y están optimizados para realizar búsquedas veloces. La relevancia de la mejora introducida es dejar obsoleta la limitación original del algoritmo en cuanto a la cantidad de dimensiones que se pueden procesar. No se ha realizado un estudio comparativo entre la velocidad de mapeo del algoritmo original (acceso directo a la celda) y el algoritmo con la mejora introducida (acceso mediante consulta de un diccionario de datos) ya que un aumento en el tiempo de mapeo objeto/celda es compensado (al menos en parte) por la drástica disminución de celdas blancas a evaluar

como último paso del algoritmo. En la figura 2, se muestra la ejecución del programa mejorado. La cantidad de celdas que requiere el algoritmo se redujo de 60 millones a 2.027 que son las necesarias para alojar todos las filas de datos y sus vecinos L_1 y L_2.

3.2. Persistencia de la relación objeto/celda en un almacenamiento secundario.

Para experimentar con los límites del algoritmo **FindAllOutsM** se creó un banco de datos artificial con nuevas tuplas utilizando la técnica descripta en el trabajo original [3] para la creación de tuplas sintéticas que reflejan la distribución de las estadísticas dentro del banco de datos NHL94. Con el objetivo de alojar solamente en memoria las celdas requeridas por el algoritmo (y así consumir menos memoria), la relación objeto/celda se persiste en un almacenamiento alternativo. Cada objeto, más allá de sus atributos numéricos utilizados para determinar si es o no outlier, tienen un atributo extra obligatorio que es una clave que identifica la tupla. Dicha clave sirve para volver a obtener los datos del objeto a demanda. Así, cuando una celda requiera todos los objetos que se encuentran en otra celda, el programa puede volver a obtener los datos necesarios para calcular las distancias objeto a objeto como lo requiere el algoritmo. Por las características propias del algoritmo la cantidad de objetos que deban ser releídos debe ser muy inferior al número total de tuplas del banco de datos.

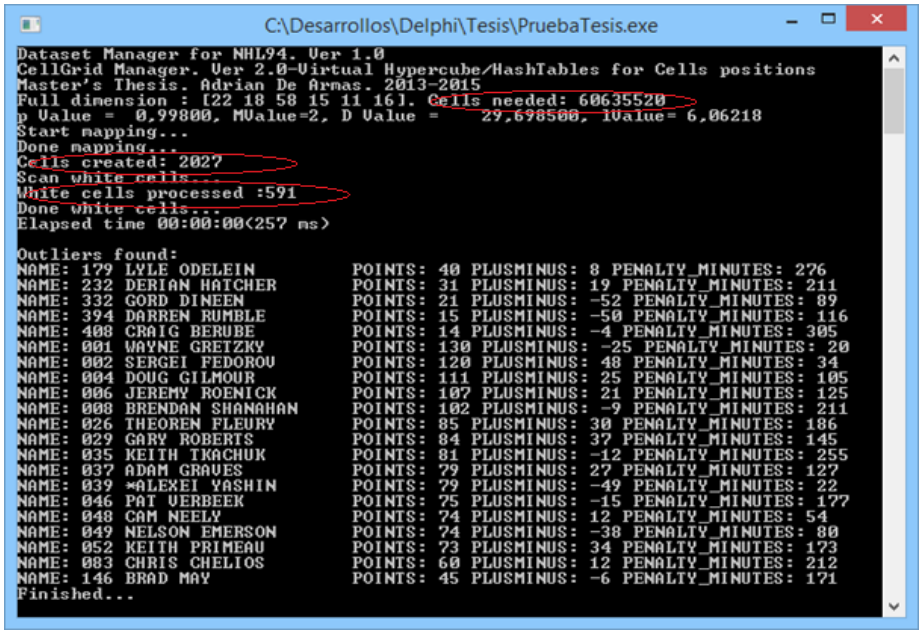

Figura 2 – FinAllOutsM con hipercubo virtual para N=871 y D=6

Las modificaciones realizadas alcanzaron sus objetivos y disminuyeron el consumo de memoria del programa, como puede observarse en la figura 3, 5 millones de objetos (y sus celdas asociadas) consumieron sólo 190 megas de memoria al realizar el proceso mientras que la misma versión del algoritmo (de 32 bits), sin esta modificación, cancela

por no poder alojar en memoria RAM las celdas, los objetos y el mapeo objeto/celda. Esta modificación extendió considerablemente la cantidad de filas que se pueden procesar ya que toda la memoria se utiliza para alojar celdas. Con el programa compilado para 64 bits, el único limitante de la cantidad de celdas a crear es la cantidad de memoria disponible en el equipo.

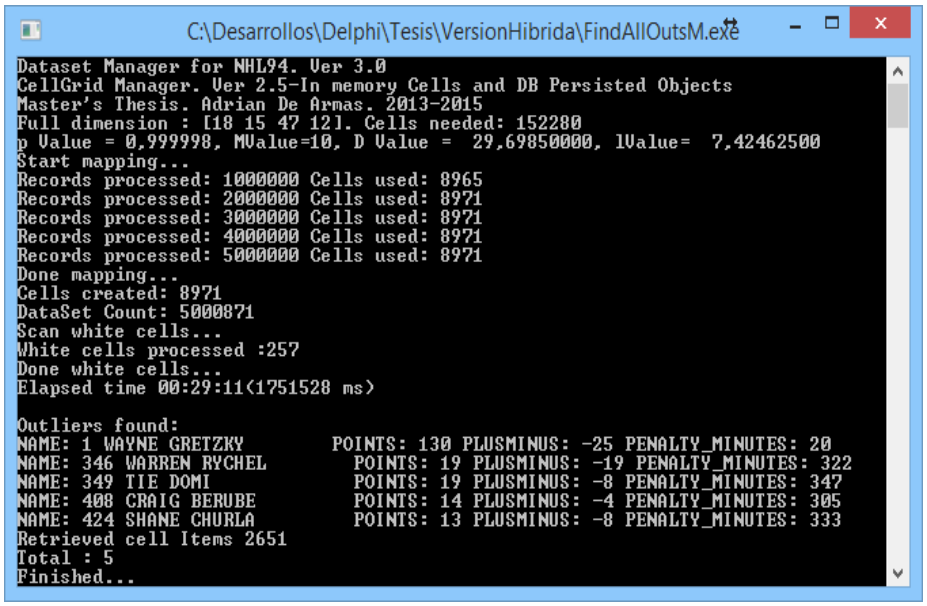

Figura 3 –**FindAllOutsM** híbrido para N=5.000.871 y D=4.

3.3. Paralelización del algoritmo

Si como medida de eficiencia de un algoritmo se tiene en cuenta qué tan bien aprovecha todos los recursos del equipo en el que se ejecuta, se introduce una nueva oportunidad de mejora: la capacidad de procesamiento que brinda un equipo con varios procesadores.

Una versión paralela del algoritmo **FindAllOutsM** sería similar a la versión no paralela, sólo se diferenciaría por el hecho de que soporta más de un hilo de ejecución concurrente, esto quiere decir, que es capaz de ejecutar simultáneamente múltiples secuencias de instrucciones. Cada secuencia de instrucciones tiene su propio control de flujo de ejecución que es independiente de los otros controles de flujo. Cada secuencia de instrucciones independiente se conoce como hilo de ejecución (o simplemente hilo).

El algoritmo **FindAllOutsM** tiene dos etapas bien diferenciadas que pueden verse beneficiadas por la paralelización. La primera etapa, donde se realiza el mapeo celda/objeto, se puede agrupar bajo el nombre de “Carga” y la segunda es la “determinación fina” de outliers.

Entre la "carga" y la "determinación fina" hay una diferencia relevante y es que la carga debe leer y actualizar una zona común de datos (el hipercubo de celdas) mientras que la determinación fina accede al mismo hipercubo pero sólo en modo lectura.

La estrategia utilizada para paralelizar la carga de objetos fue dividir el banco de datos en lotes que cada hilo podría procesar individualmente mientras que la estrategia utilizada para paralelizar la determinación fina de outliers fue que cada hilo tomara una celda blanca y procesara la celda junto con su entorno para determinar si todos los objetos de la celda son outliers o no. Cada hilo de ejecución solicita que se le entregue una celda blanca que procesar. La ejecución termina cuando el proceso que administra todos los hilos no tiene más celdas que entregar.

Una vez introducida la mejora, se experimentó con bancos de datos de 2.500.000, 5.000.000 y 10.000.000 de tuplas y con 3, 4, 5 y 6 atributos. Los resultados de la ejecución del algoritmo para $p = 0{,}9999$ y $D = 29{,}6985$ pueden observarse en las tablas 1 y 2.

Versión todo en memoria para p=0,9999 y D=29.6985						
N	3D		4D		5D	
	Original	Paralela	Original	Paralela	Original	Paralela
2.500.000	1m 28s	1m 4s	1m 31s	1m 5s	1m 46s	1m 13s
5.000.000	4m 41s	2m 33s	4m 27s	2m 21s	4m 56s	2m 38s
10.000.000	14m 13s	5m 31s	14m 38s	5m 10s	15m 28s	5m 42s

Tabla 1 – Experimentos ejecutados **FindAllOutsM** en memoria original y paralelo

Versión híbrida para p=0,9999 y D=29.6985						
N	3D		4D		5D	
	Original	Paralela	Original	Paralela	Original	Paralela
2.500.000	34m 54s	13m 5s	27m 22s	10m 43s	11m 48s	8m 45s
5.000.000	47m 9s	20m 39s	38m 59s	16m 48s	31m 32s	13m 48s
10.000.000	1h 49m 4s	30m 6s	1h 6m 47s	34m 40	1h 17m 35s	31m 45s

Tabla 2 – Experimentos ejecutados **FindAllOutsM** híbrido original y paralelo

Para 6 atributos los parámetros utilizados en los experimentos anteriores no permitían desarrollar el potencial del algoritmo **FindAllOutsM**. Se observó que se generaron, producto del mapeo celda/objetos, todas celdas blancas, ninguna rosa ni roja. Esto se explica por la cantidad de atributos a procesar que influye en la formación del hipercubo con más de 60 millones de celdas, donde se tienen que alojar todas las tuplas. Como resultado se obtienen celdas blancas con objetos cuya etiqueta de "outlier" (o no) es determinada por un proceso de comparación objeto a objeto. El algoritmo **FindAllOutsM** se comporta como una determinación de outliers por fuerza bruta más allá que la comprobación se restringe a sus vecinos L_2. En la tabla 3 pueden observarse los experimentos para 6 atributos con nuevos parámetros.

Estos experimentos permiten demostrar la relevancia de seleccionar correctamente los parámetros de cada ejecución del algoritmo con el fin de aprovechar las ventajas que brinda **FindAllOutsM** como así también, para obtener resultados relevantes.Los experimentos permitieron confirmar que el algoritmo puede verse beneficiado con el uso de varios procesadores a medida que la cantidad de tuplas crece.

p=0,999995 y D=44.6985						
N	6D memoria			6D Híbrida		
	Normal	Paralela	Diferencia	Normal	Paralela	Diferencia
2.500.000	2m 26s	1m 51s	23,97%	13m 31s	9m 58s	26,26%
5.000.000	5m 25s	3m 4s	43,38%	23m 12s	15m 55s	31,39%
10.000.000	11m 15s	5m 37s	50,07%	42m 27s	27m 19s	35,65%

Tabla 3 – Experimentos ejecutados para **FindAllOutsM** original y paralelo

En la figura 4 se observa cómo el uso de varios procesadores pasa inadvertido para cantidades pequeñas de tuplas ya que el tiempo ganado por la ejecución en paralelo se ve compensado por la complejidad interna del programa dedicado a administrar los distintos hilos de ejecución. A medida que la cantidad de tuplas aumenta, los beneficios de paralelizar son más notorios ya que permite suavizar el incremento de tiempo requerido hasta arrojar resultados.

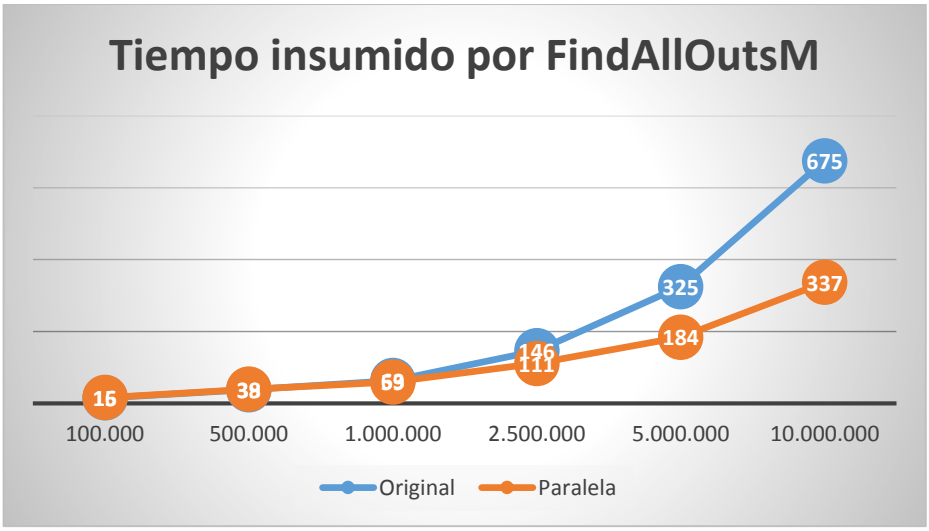

Figura 4 –**FindAllOutsM** para 6 atributos, p=0,999995 y D=44,6985

3.4. Detección probabilística de outliers

El algoritmo **FindAllOutsM** tiene por característica detectar el 100% de los outliers basados en distancia. Existen situaciones dónde es más importante obtener un rápido resultado que una eficacia del 100%. La determinación probabilística de outliers se basa en obtener probabilísticamente cuántos elementos aporta una celda en particular al vecindario más cercano de un objeto cualquiera. Como prueba de concepto la probabilidad se calcula con una simple regla de tres simple y se asume que los objetos asociados a una celda se encuentran uniformemente distribuidos.

Por cada celda contra la cual hay que calcular la distancia objeto a objeto, se calcula la distancia máxima y mínima que puede tener el objeto a analizar contra cualquier objeto perteneciente a esa celda, luego, utilizando la regla de tres simple, el porcentaje de objetos que esa celda puede aportar de vecinos más cercanos queda expresado:

$$porcentaje = (D - DistMin)/(DistMax - DistMin)$$

Donde D es el valor de la distancia obtenida como parámetro para la ejecución del algoritmo. Una vez calculado ese porcentaje, la cantidad de elementos que aporta la celda al objeto se calcula con $vecinos = Cantidad\ objetos * porcentaje$, siendo $Cantidad\ objetos$ los objetos asociados a la celda L_2 procesada. Para maximizar la posibilidad de obtener resultados relevantes esta técnica sólo es utilizada cada vez que la celda L_2 a analizar sea una celda roja por dos motivos:

1. Cuánto más objetos tenga asociado la celda roja, más tiempo se requiere para recuperar los datos de los objetos asociados
2. Cuánto más elementos tenga la celda, mayor probabilidad de obtener resultados que se asemejen a la realidad (los objetos se distribuyen uniformemente en la celda)

Las tablas 4 y 5 muestran el resultado de los experimentos. La versión "híbrida" es la que persiste en una memoria alternativa la relación celda/objeto. Esta versión es la que más se beneficia por la determinación probabilística de outliers por no tener que recuperar objetos desde el disco. Se dividieron los experimentos para utilizar distintos parámetros según la cantidad de atributos. Las columnas FP y NE se refieren a los falsos positivos detectados y los outliers no encontrados respectivamente.

p=0,9999 d=29.6985										
N	3D					4D				
	Hibrida	Prob	OUTLIERS	FP	NE	Híbrida	Prob	OUTLIERS	FP	NF
10.000.000	34m 24s	3m 2s	216	71	0	27m 55s	10m 37s	230	44	0

Tabla 4 – Experimentos ejecutados para **FindAllOutsM** híbrida y probabilística

p=0,999995 d=44.6985										
N	5D					6D				
	Híbrida	Prob	OUTLIERS	FP	NE	Híbrida	Prob	OUTLIERS	FP	NF
2.500.000	3M 23S	25S	3	0	0	3m 33s	23s	3	0	0
5.000.000	8m 51s	42s	3	8	0	3m 17s	2m 45s	3	0	0
10.000.000	20m 52s	2m 41s	3	15	0	5m 55	5m	3	0	0

Tabla 5 – Experimentos ejecutados para **FindAllOutsM** híbrida y probabilística

Del análisis del resultado de los experimentos puede concluirse que el porcentaje de mejora en tiempo obtenido por el algoritmo **FindAllOutsM** (en detrimento de la eficacia del mismo) siempre ha superado el porcentaje de falsos positivos detectados. Durante todos los experimentos nunca se registró una situación en donde el algoritmo no detectase algún outlier real (aunque es una posibilidad cierta), la única anomalía registrada fue la detección de falsos positivos como se puede observar en la figura 5.

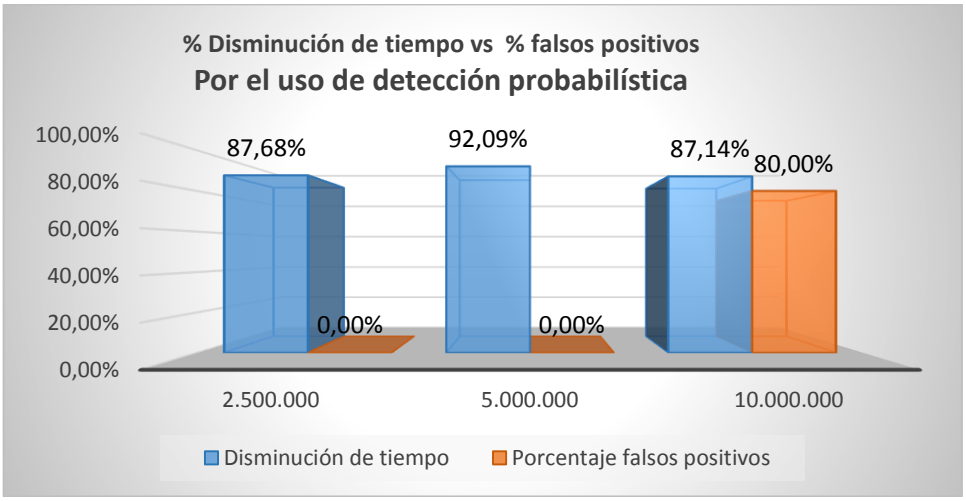

Figura 5 –FindAllOutsM para 5 atributos, p=0,999995, D=44,6985, N=10.000.000

4. Conclusiones

En este trabajo se implementó, con mejoras el algoritmo de aproximación por celdas para detección de outliers FindAllOutsM [3]. Disponer de algoritmos capaces de ser utilizados en estos escenarios es importante ya que el tamaño del conjunto de datos y el número de dimensiones han comprobado ser obstáculos clave para el análisis de los datos [4].

La implementación del algoritmo fue dotada de características esenciales para lograr el objetivo: se eliminó la restricción original sobre la cantidad de atributos y sus distintas versiones pueden ser tenidas en cuentas para distintos escenarios.

La restricción inicial del método por la cual se podían procesar 4 (a lo sumo 5) atributos se eliminó virtualizando el hipercubo de celdas requerido para el mapeo de celdas y objetos. Mejoras como la paralelización del algoritmo o la detección probabilística de outliers mejoraron el rendimiento de ejecución. Las versiones híbridas habilitan la utilización del método en una variedad más amplia de equipos porque requieren menos memoria RAM al utilizar almacenamientos alternativos como un disco duro.

Superadas las limitaciones iniciales junto con las modificaciones realizadas que mejoran el rendimiento del algoritmo, habilitan a que **FindAllOutsM** pueda ser considerado a la hora de determinar qué outliers basados en distancia se encuentran en un banco de datos de gran cantidad objetos sin importar la cantidad de atributos a procesar cumpliendo, de esta manera, el objetivo incial de esta propuesta.

Referencias

[1] KNORR, E. M (2002)**.** *Outliers and Data Mining: Finding Exceptions in Data* [En linea]. Tesis doctoral. Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada. Consulta realizada el 13/08/2013. https://www.cs.ubc.ca/grads/resources/thesis/May02/Ed_Knorr.pdf

[2] HAWKINS D**.** (1980). *Identification of Outliers (Monographs on Statistics and Applied Probability) vol 3.* Chapman and Hall, London.

[3] KNORR E.M.;NG R.T.(1998). *Algorithms for Mining Distance-Based Outliers in Large Datasets* [En linea]. Proceedings of 24rd International Conference on Very Large Data Bases August 24-27, 1998, New York, New York, USA. http://www.vldb.org/dblp/db/conf/vldb/KnorrN98.html. Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada. Consulta realizada el 13/08/2013. www.vldb.org/conf/1998/p392.pdf

[4] HAN J.;KAMBER M. (2006). *Data Mining – Concepts and techniques*. 2a edición. The Morgan Kaufmann Series in Data Management Systems. Elsevier Inc, USA.

[5] DAVIES, L.;GATHER, U.(1993) *The identification of multiple outliers (with discussion)*. Journal ofthe American Statistical Association 88. Taylor & Francis Group, USA.

[6] BARNET V.;LEWIS T.(1994). *Outliers in Statical Data.* John Wiley, 3a edición. Elsevier Inc, USA.

[7] LOZANO E.;ACUÑA E. (2005). *Parallel Algorithms for Distance-Based and Density-Based Outliers* [En linea]. IEEE International Conference on Data Mining (ICDM). Consulta realizada el 26/06/2014. academic.uprm.edu/eacuna/elioedgarieee.pdf

[8] BILLOR N.;KIRAL G. (2008). *A comparison of Multiple Outlier Detection Methods for Regression Data*. Comunication in Statistics – Simulation and Computation. Taylor & Francis Group, LLC. USA.

[9] BREUNIG M.; KRIEGEL H.; NG R.; SANDER J.(2000). *LOF: Identifying Density-Based local Outliers* [En linea]. ACM SIGMOD Conference 2000. USA. Consulta realizada en 10/03/2014. www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf

[10] KRIEGEL H.;SCHUBERT M;ZIMEK A.(2008). *Angle-Based Outlier Detection in High-dimensional Data* [En linea]. Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, Pages 444-452, ACM New

York, USA. http://dl.acm.org/citation.cfm?id=1401946. Department of Computer Science & Database Systems Group - Ludwig-Maximilians-Universität München. Consulta realizada el 21/08/2013. www.dbs.ifi.lmu.de/~zimek/.../KDD08-ABOD.pdf
[11] PAPADIMITRIOU S.;KITAGAWA H.;GIBBONS P.; FALOUTSOS C. (2003)*.LOCI: Fast Outlier Detection using the Local Correlation Integral* [En linea]. International Conference on Data Engineering (ICDE). Consulta realizada el 28/06/2014. http://www.informedia.cs.cmu.edu/documents/loci_icde03.pdf
[12] PHAM N.; PAGH R.(2012). *A Near-linear Time Approximation Algorithm for Angle-based Outlier Detection in High-dimensional Data* [En linea]. Proceeding KDD 12th Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, Pages 877-885, ACM New York, NY, USA. http://dl.acm.org/citation.cfm?id=2339669. IT University of Copenhagen, Dinamarca. Consulta realizada el 12/09/2013. www.itu.dk/people/pagh/papers/outlier.pdf
[13] AGGARWAL C.(2013). *Outliers Analysis*. Springer, IBM T.J. Watson Research Center,Yorktown Heights, New Work, USA.
[14] DE ARMAS A. (2015). *Detección de outliers en grandes bases de datos*. Trabajo Final. Maestría en Tecnología Informática y de Comunicaciones. Universidad Argentina de la empresa, Facultad de Ingeniería y Ciencias Exactas. Buenos Aires. Argentina.