

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“COMPARACIÓN CUANTITATIVA DE DESEMPEÑO
ENTRE ALGORITMOS SECUENCIALES Y ALGORITMOS
PARALELOS DE ANÁLISIS FORMAL DE CONCEPTOS”

JORGE ARIEL DÍAZ MATTE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Víctor Codocedo
Profesor Correferente: Erika Rosas Oliva

Diciembre - 2021

DEDICATORIA

Dedico este trabajo a mis padres, a mi polola y a mis hermanos, por su apoyo durante todos estos años, por motivarme a ser mejor persona y profesional, por consolarme en los momentos malos, celebrar conmigo en los momentos felices y por infinitas cosas más.

A toda mi familia, mis abuelos, tíos, primos y sobrinos por su preocupación, ayuda y compañía en los momentos importantes.

A mis profesores de colegio, quienes, con su paciencia y talento, me educaron y permitieron desarrollar mis capacidades para poder seguir estudiando en la universidad. A mis profesores del departamento que me introdujeron en el mundo de la informática y siempre se preocuparon de que fuera un mejor profesional.

Por último, a todos los estudiantes o futuros estudiantes que, durante su educación básica y media, estudiaron en colegios promedio y sin muchas expectativas, les aseguro que las notas y la prioridad no los definen como profesionales, si no, su actitud ante el constante perfeccionamiento, auto aprendizaje y trabajo en comunidad.

AGRADECIMIENTOS

Agradezco profundamente a mis padres, Nanette y Ariel, por su valentía y amor que me han brindado a lo largo de todos estos años, por su entrega, por su esfuerzo y sacrificio, por sus aciertos y errores que me permitieron llegar hasta este momento. A mi polola Raiza, por su amor, por su entrega, por su compañía, por enseñarme a valorarme, por sus consejos, por su aliento para no rendirme cuando todo parece ir mal y me siento derrotado. A mis hermanos por su preocupación y compañía, por permitirme ayudarlos y enseñarles cosas, por pasar tiempo juntos y siempre considerarnos.

A mi familia numerosa, por todas esas festividades, por todas esas vacaciones, paseos o simplemente visitas que siempre fueron tan alegres y cómodas. A mi abuela Eliana por cuidarme desde siempre, quererme y aguantarme. A mis abuelos paternos, Norma y Melquicedec, por todos esos veranos juntos los tres, por todos esos fines de semanas donde siempre estaban atentos de mí. A Martín y Carola por siempre considerarme, escucharme y ayudarme desde siempre, tanto en mi vida personal como en la profesional como fue en esa inolvidable practica de trabajo. A mis primos que siempre han estado conmigo Sabrina, Nicolás, Anely, por siempre estar en todas esas reuniones familiares, desde esas grabaciones en VHS hasta hoy en día. A todos mis tíos por siempre haber estado ahí y ayudarme desinteresadamente. A la tía Luci por no dudar un segundo en apoyarme cuando decidí entrar a la USM.

A los profesores Víctor Codocedo y Erika Rosas, por todo su apoyo en el desarrollo de esta memoria, por sus consejos y ayuda en el proceso de postulación a magister, y todas las oportunidades que me han dado para desarrollarme como profesional. A la profesora Natalia González y al profesor Claudio Lobos por permitirme iniciar como ayudante y ganar confianza para lograr mis metas. Al profesor José Luis Martí por su ayuda incondicional a lo largo de la carrera, considerarme y brindarme las oportunidades para hacer los talleres. A todos estos profesores les agradezco profundamente por hacerme sentir parte de una comunidad que no discrimina y se preocupa por el real desarrollo y formación sus estudiantes.

A mis amigos de la universidad: A los Apyr Ñaki, Isidora, Sebastián, Luis, Benjamín, Camilo y Guillermo por las juntas y las partidas. A todos los Peryton Francisco, Gabriel, Jorge, Sebastián, Rodrigo y Michell, por todo el trabajo que hemos hecho juntos, la ayuda y paciencia. Al Sebastián Godínez por todas las veces que fuimos equipo juntos. A la Monse por sus consejos y partidas de taca en primer año. Al Fabián por todas esas conversaciones de Star Wars y sus múltiples juntas. A mis amigos de colegio: Bastián por todos esos recuerdos, juntas, caminatas, ventas de helados y compras en la distribuidora. A Ignacio, Martín y Nicolás por la existencia del círculo cerrado, todos esos recuerdos, salidas y capeos. A mis profesores de colegio que siempre se me apoyaron para mejorar y ser mejor estudiante, a la profesora Cecilia, Beatriz, Antonio, César.

RESUMEN

Resumen— El Análisis Formal de Conceptos, es una teoría matemática presentada por Rudolf Willie en 1982 y que ha sido continuamente estudiada, logrando avances hasta el día de hoy. En este documento se presenta un estudio respecto al desempeño de diferentes algoritmos de Análisis Formal de Conceptos, comparando los resultados obtenidos por algoritmos secuenciales y algoritmos paralelos. Para ello, se investigó respecto el Análisis Formal de Conceptos y el funcionamiento de los algoritmos, lo que permitió crear una librería en la que se implementó los algoritmos que se seleccionaron para este estudio. A partir de los resultados se evidencia la ventaja que tiene el algoritmo NextClosure frente a la mayoría de los algoritmos en las diferentes pruebas que se hicieron, y los buenos desempeños obtenidos por la propuesta del algoritmo Paraléctico, que combina los enfoques Breath first y Deepth first junto con la programación paralela.

Palabras Clave— Análisis Formal de Conceptos; Conceptos Formales; Algoritmos Secuenciales; Algoritmos Paralelos; Desempeño.

ABSTRACT

Abstract— The Formal Concept Analysis, is a mathematical theory presented by Rudolf Wille in 1982, which has been continuously studied making advances to this day. This document presents a study regarding the performance of different algorithms of Formal Concepts Analysis, comparing the results obtained by sequential algorithms and parallel algorithms. For this reason, it was investigated on the Formal Concept Analysis and the operation of the algorithms, which allowed the creation of a library in which the algorithms that were selected for this study were implemented. From the results, the advantage of the algorithm NextClosure is evident compared to most of the algorithms in the different tests that were made, and the good performances obtained by the Parallelectical algorithms that combine the approaches breadth first and depth first with parallel computing.

Keywords— Formal Concept Analysis; Formal Concept; Sequential Algorithm; Parallel Algorithm; Performance.

GLOSARIO

Aquí se deben colocar las siglas mencionadas en el trabajo y su explicación, por orden alfabético. Por ejemplo:

DI: Departamento de Informática.

UTFSM: Universidad Técnica Federico Santa María.

AFC: Análisis Formal de Conceptos.

FCA: Formal Concept Analysis.

GPU: Graphics Processing Unit.

PPPL: Primera Propuesta Paralelización Léctica.

SPPL: Segunda Propuesta Paralelización Léctica .

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABLAS	XI
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Contexto	2
1.2 Definición	3
1.3 Objetivos	3
1.3.1 Objetivo General	3
1.3.2 Objetivos Específicos	3
CAPÍTULO 2: MARCO CONCEPTUAL	4
2.1 Análisis Formal de Conceptos	4
2.1.1 Contexto Formal	4
2.1.2 Operadores de Derivación	5
2.1.3 Conceptos Formales	6
2.1.4 Jerarquía de Conceptos	7
2.1.5 Retículos	8
2.2 Clausuras	10
2.2.1 Definición	11
2.2.2 Operadores de Clausura	11
2.3 Programación Paralela	12
2.3.1 Patrones de Programación Paralela	13
2.3.2 Entornos de ejecución para la programación paralela	14
2.3.3 Balance de Carga	15
2.4 Tipos de Enumeraciones	15
2.4.1 Enumeración Lexicográfica	16
2.4.2 Enumeración Léctica	16
2.5 Algoritmos Secuenciales	17
2.5.1 Algoritmo NextClosure	17
2.5.2 Algoritmo Apriori	19
2.6 Algoritmos Paralelos o Distribuidos	21
2.6.1 Paralelización del Algoritmo NextClosure	21
2.6.2 Paralelización del Algoritmo Apriori	21

2.6.3 Paralelización de la Enumeración Léctica	23
2.6.4 Otros Algoritmos	26
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	28
3.1 Algoritmos Seleccionados	28
3.2 Datasets Seleccionados	29
3.3 Diseño Librería	29
3.3.1 Utilidades de la Librería	30
3.4 Entorno de Ejecución	30
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	31
4.1 Implementación de Librería	31
4.2 Definición Campos Resultados	32
4.3 Resultados Obtenidos PPPL	32
4.4 Resultados Generales	35
4.5 Análisis resultados	38
4.6 Estudio Caso Dataset Servo	41
4.6.1 Dataset Servo	43
4.6.2 Dataset Servo Invertido	44
4.6.3 Dataset Adult	46
4.6.4 Dataset Adult Invertido	47
4.7 Análisis Resultados Caso Dataset Servo	49
CAPÍTULO 5: CONCLUSIONES	51
5.1 Sobre el Trabajo Realizado	51
5.2 Sobre los Resultados Obtenidos	51
5.3 Sobre el Cumplimiento de los Objetivos	53
5.4 Trabajo a Futuro	53
5.5 Palabras Finales	54
ANEXOS	56
5.1 Algoritmo PPPL	56
5.1.1 Resultados Dataset Adult	56
5.1.2 Resultados Dataset Car	58
5.1.3 Resultados Dataset Credit	60
5.1.4 Resultados Dataset Flight	62
5.1.5 Resultados Dataset Horse	64
5.1.6 Resultados Dataset Mushroom	66
5.1.7 Resultados Dataset Servo	68
5.2 Comparación General	70
5.2.1 Dataset Adult	70
5.2.2 Dataset Car	72
5.2.3 Dataset Credit	74
5.2.4 Dataset Flight	76

5.2.5 Dataset Horse	78
5.2.6 Dataset Mushroom	80
5.2.7 Dataset Servo	82
REFERENCIAS BIBLIOGRÁFICAS	84

ÍNDICE DE FIGURAS

1	Ejemplo Diagrama de Hasse	2
2	Diagrama de Hasse	10
3	Patrón Maestro/Esclavo	13
4	Patrón Dividir y Conquistar	14
5	Ejemplo división de trabajo SPPL	26
6	Tiempo de ejecución en PPPL por Tamaño de Carga en Diferentes Contextos . .	33
7	Clausuras calculadas en PPPL por tamaño de carga en diferentes contextos . . .	34
8	Memoria utilizada en PPPL por tamaño de carga en diferentes contextos	34
9	Resultados tiempos de ejecución algoritmos a diferentes contextos	36
10	Resultados clausuras calculadas algoritmos a diferentes contextos	37
11	Resultados memoria utilizada algoritmos a diferentes contextos	37
12	Tiempo de ejecución por sección de los algoritmos en contexto Servo.	43
13	Candidatos revisados por nivel en Apriori en contexto Servo.	43
14	Clausuras calculadas por concepto formal en NextClosure en contexto Servo. .	44
15	Tiempo de ejecución de los algoritmos en contexto Servo Invertido.	44
16	Candidatos revisados por nivel en Apriori en contexto Servo Invertido.	45
17	Clausuras calculadas por concepto formal en NextClosure en contexto Servo Invertido.	45
18	Tiempo de ejecución de los algoritmos en contexto Adult.	46
19	Candidatos revisados por nivel en Apriori en contexto Adult.	46
20	Clausuras calculadas por concepto formal en NextClosure en contexto Adult. .	47
21	Tiempo de ejecución de los algoritmos en contexto Adult Invertido.	47
22	Candidatos Revisados por nivel en Apriori en contexto Adult Invertido.	48

23 Clausuras calculadas por concepto formal en NextClosure en Adult Invertido. . .	48
24 Resultados de Tiempo PPPL Adult	56
25 Resultados Número de Clausuras Calculadas PPL Adult	57
26 Resultados Memoria Utilizada PPPL Adult	57
27 Resultados de Tiempo PPPL Car	58
28 Resultados Número de Clausuras Calculadas PPPL Car	59
29 Resultados Memoria Utilizada PPPL Car	59
30 Resultados de Tiempo PPPL Credit	60
31 Resultados Número de Clausuras Calculadas PPPL Credit	61
32 Resultados Memoria Utilizada PPPL Credit	61
33 Resultados de Tiempo PPPL Flight	62
34 Resultados Número de Clausuras Calculadas PPPL Flight	63
35 Resultados Memoria Utilizada PPPL Flight	63
36 Resultados de Tiempo PPPL Horse	64
37 Resultados Número de Clausuras Calculadas PPPL Horse	65
38 Resultados Memoria Utilizada PPPL Horse	65
39 Resultados de Tiempo PPPL Mushroom	66
40 Resultados de Número de Clausuras Calculadas PPPL Mushroom	67
41 Resultados de uso de Memoria PPPL Mushroom	67
42 Resultados de Tiempo PPPL Servo	68
43 Resultados Número de Clausuras Calculadas PPPL Servo	69
44 Resultados Memoria Utilizada PPPL Servo	69
45 Resultados Tiempos Dataset Adult	70
46 Resultados Número de Clausuras Calculadas Adult	71

47 Resultados Memoria Utilizada Adult	71
48 Resultados Tiempos Dataset Car	72
49 Resultados Número de Clausuras Calculadas Car	73
50 Resultados Memoria Utilizada Car	73
51 Resultados Tiempos Dataset Credit	74
52 Resultados Número de Clausuras Calculadas Credit	75
53 Resultados Memoria Utilizada Credit	75
54 Resultados Tiempos Dataset Flight	76
55 Resultados Número de Clausuras Calculadas Flight	77
56 Resultados Memoria Utilizada Flight	77
57 Resultados Tiempos Dataset Horse	78
58 Resultados Número de Clausuras Calculadas Horse	79
59 Resultados Memoria Utilizada Horse	79
60 Resultados Tiempos Mushroom	80
61 Resultados Número de Clausuras Calculadas Mushroom	81
62 Resultados Memoria Utilizada Mushroom	81
63 Resultados Tiempos Servo	82
64 Resultados Número de Clausuras Calculadas Servo	83
65 Resultados Memoria Utilizada Servo	83

ÍNDICE DE TABLAS

1 Contexto Formal de Animales Vertebrados.	5
2 Contexto Formal números divisores	9
3 Algoritmos Secuenciales Seleccionados	28

4	Algoritmos Paralelos Seleccionados	29
5	Datasets Seleccionados	29
6	Tiempos de ejecución algoritmo PPPL a diferentes cargas.	32
7	Número de clausuras calculadas algoritmo PPPL a diferentes cargas.	33
8	Memoria utilizada algoritmo PPPL a diferentes cargas.	33
9	Resultados tiempos de ejecución por algoritmos.	35
10	Resultados número de clausuras calculadas.	35
11	Resultados memoria utilizada.	36
12	Datasets seleccionados	41
13	Resultados tiempos de ejecución Apriori y NextClosure	42
14	Resultados clausuras calculadas Apriori y NextClosure	42
15	Resultados memoria utilizada Apriori y NextClosure	42
16	Resultados Algoritmo PPPL Dataset Adult.	56
17	Resultados Algoritmo PPPL Dataset Car.	58
18	Resultados Algoritmo PPPL Dataset Credit.	60
19	Resultados Algoritmo PPPL Dataset Flight.	62
20	Resultados Algoritmo PPPL Dataset Horse.	64
21	Resultados Algoritmo PPPL Dataset Mushroom.	66
22	Resultados Algoritmo PPPL Dataset Servo.	68
23	Resultados Algoritmos Dataset Adult.	70
24	Resultados Algoritmos Dataset Car.	72
25	Resultados Algoritmos Dataset Credit.	74
26	Resultados Algoritmos Dataset Flight.	76
27	Resultados Algoritmos Dataset Horse.	78

28 Resultados Algoritmos Dataset Mushroom.	80
29 Resultados Algoritmos Dataset Servo.	82

INTRODUCCIÓN

El Análisis Formal de Conceptos fue propuesto por Rudolf Wille en los años 80's y actualmente es ocupado como una herramienta para Data Mining y Knowledge Discovery [Ganter y Wille, 1999b]. En esta área de estudio se utilizan diversos algoritmos que buscan cumplir con diferentes tareas pero el tiempo que tardan en ejecutarse estos algoritmos es un problema que motiva a seguir estudiando esta materia.

Por otro lado, el avance tecnológico en relación a arquitecturas multinúcleos a motivado al desarrollo de algoritmos y tecnologías paralelas o distribuidas que buscan disminuir el tiempo de ejecución repartiendo una tarea entre varios procesos o computadores [Li y Zhang, 2018].

En esta memoria se compara el desempeño de algoritmos de Análisis Formal de Conceptos secuenciales y paralelos a partir de una selección, con el objetivo de evaluar cuáles son los enfoques que obtienen mejores resultados. Para esto, se seleccionaron cinco algoritmos, dos de ellos son secuenciales y el resto corresponden a algoritmos paralelos. Además, dichos algoritmos se diferencian por utilizar enfoques *breath-first* o *depth-first* para explorar el espacio de búsqueda. Esta selección de algoritmos fueron ejecutados en una selección de siete *datasets* que poseen diferentes características.

Nuestros resultados mostraron que la paralelización de algoritmos *depth-first* obtuvo mejor desempeño que aquellos que utilizan *breath-first*. Esto se atribuye a la ventaja algorítmica que brinda en la enumeración léctica a través del uso del test canónico que evita el calculo de clausuras redundantes.

Esta memoria está estructurada de la siguiente forma:

- **Capítulo 1:** En este se define el problema a tratar, donde se contextualiza el estudio, se presentan los objetivos y el alcance del trabajo.
- **Capítulo 2:** Se presenta la base conceptual para el entendimiento del trabajo del documento y se explican algunos de los algoritmos existentes para el Análisis Formal de Conceptos.
- **Capítulo 3:** Se presentan los algoritmos seleccionados para el estudio, junto con los *datasets* elegidos para realizar las pruebas y el entorno de ejecución donde serán probados.
- **Capítulo 4:** Se presentan los resultados de las pruebas de los algoritmos en los diferentes *datasets*, se comparan estos resultados y se discute su comportamiento.
- **Conclusiones:** Se presentan las principales inquietudes resultantes del trabajo realizado, la verificación de los objetivos planteados y el trabajo a futuro.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

Formal Concept Analysis (FCA) o Análisis Formal de Conceptos (AFC) en español, es una teoría matemática presentada por primera vez en 1982 por Rudolf Wille, que busca representar una estructura de jerarquía entre los objetos y sus propiedades presentes en un Contexto Formal [Min y Kim, 2019].

Este contexto está compuesto por un conjunto de objetos, un conjunto de atributos, una relación binaria entre ellos, que son comúnmente representados como tablas. En este escenario, FCA trabaja para encontrar los llamados Conceptos Formales, que es un par formado por un conjunto de objetos denominado extensión y un conjunto de atributos nombrado intención [Ganter *et al.*, 2016]. De esta manera, un Concepto Formal posee la particularidad de que la extensión está compuesta por todos los objetos que compartan los atributos de la intención, y, a su vez, la intención contiene a todos los atributos presentes en todos los objetos de la extensión. A los conjuntos con esta propiedad se les suele denominar conjuntos cerrados, y el proceso para verificar que un conjunto de objetos o atributos cumple con esto se llama cálculo de clausuras.

En esta teoría, Wille utiliza una estructura matemática llamada *Lattice* o Reticulo en español, que permite ordenar los Conceptos Formales presentes en un Contexto Formal y presentarlos en un Diagrama de Hasse [Cimiano, 2005]. En la figura 1 se puede observar un ejemplo de diagrama de Hasse de álgebra de grupos. Estos conceptos pueden estar parcialmente ordenados, generando una jerarquización donde existan algunos conceptos contenidos en otros, creando una relación de subconcepto y superconcepto.

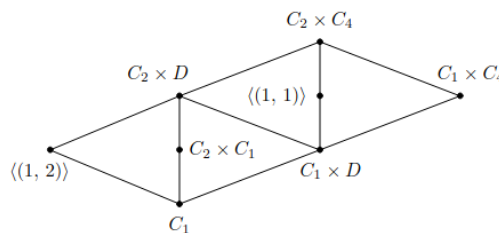


Figura 1: Ejemplo Diagrama de Hasse
Fuente: [Vojtschovsky, 2003]

1.2. Definición

Para la tarea en la que trabaja *Formal Concept Analysis*, debe resolver un problema que es NP-Completo. Dado el conjunto de ejemplos u objetos y un conjunto de atributos, para encontrar los conceptos formales, se debe revisar exhaustivamente cualquier combinación de ejemplos y atributos, y, revisar si dicha mezcla cumple con los requisitos para ser denominada un concepto formal.

Por lo anterior, el tiempo que tardan los algoritmos de FCA en completar este proceso crece exponencialmente según aumenta la cantidad de datos sobre los cuales se trabaja. Este problema o desventaja en los algoritmos de Análisis Formal de Conceptos puede ser tratado desde heurísticas, optimizaciones o paralización de tareas.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo de esta memoria es analizar y comparar cuantitativamente algoritmos secuenciales y algoritmos paralelos de Análisis Formal de Conceptos en relación con sus tiempos de ejecución, uso de memoria y cálculos de clausuras.

1.3.2. Objetivos Específicos

Los objetivos específicos de tipo SMART se representan en base a tres puntos:

- Recolectar y analizar algoritmos existentes para el Análisis Formal de Conceptos, investigando su posible paralelización.
- Diseñar e implementar una librería que esté compuesta por una selección de algoritmos de Análisis Formal de Conceptos que se puedan paralelizar.
- Comparar el desempeño en tiempo de ejecución entre algoritmos secuenciales y algoritmos paralelos de FCA utilizando la librería implementada.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Análisis Formal de Conceptos

El Análisis Formal de Conceptos o *Formal Concept Analysis* (FCA) en inglés, fue propuesto por Rudolf Wille en los años 80's [Ganter y Wille, 2012]. Esta teoría, es ocupada como una herramienta para *Data Mining* y *Knowledge Discovery* [Liu y Wang, 2013]. FCA trabaja sobre un *Contexto Formal*, que es utilizando como representación para los datos. En dicho contexto, se busca identificar los Conceptos Formales utilizando operaciones matemáticas de conjuntos y una estructura de *lattice conceptual* [Ganter y Wille, 1999a].

2.1.1. Contexto Formal

El Análisis Formal de Conceptos busca encontrar conceptos o categorías dentro de conjunto de información. Esta información está representada en lo que se define como Contexto Formal, que busca separar los datos que contiene en dos categorías: Objetos y sus atributos. En un contexto formal, se puede observar los objetos de ejemplo existentes, y los posibles atributos presentes en dichos ejemplos. Además, nos permite saber si un elemento en particular posee cierto atributo. Esta relación debe ser binaria, pero existen algunas herramientas llamadas *Pattern Structures* que permiten convivir, en un mismo contexto, datos de tipo multivariados, como por ejemplo valores categóricos, numéricos (discretos o continuos) [Kaytoue et al., 2015]. Sin embargo, aquellos algoritmos que utilizan estructuras de patrones, trabajan igualmente con relaciones binarias, solo que de manera indirecta o no trivial.

Definición 1 (Contexto Formal) (G, M, I) está compuesto por dos conjuntos, G y M , y una relación binaria $I \subseteq G \times M$. Si $g \in G$ y $m \in M$ están relacionados, entonces el par (g, m) está en I . Lo anterior sería leído como "el objeto g tiene el atributo m " [Qi et al., 2011].

Cada uno de los pares presentes en la relación binaria I , pueden ser representados como una cruz en una tabla. En la tabla 1, se observa un extracto de una ejemplificación de contexto formal para un conjunto de datos de animales vertebrados.

El conjunto G estaría compuesto por todos los diferentes ejemplos u objetos de animales vertebrados, que, en este caso, serían $G := \{AMP, RPT, AVS, PSC, CYL, MML, MRP, LPT\}$. Por otro lado, el conjunto de atributos M está compuesto por $M := \{TER, CBLD, FUR, PRS\}$. Como se observa en la tabla 1, la intersección de la casilla de la fila AMP con columna TER,

se encuentra marcada con una $\times$, evidenciando que el par (AMP,TER) está en I . Esto implica que los elementos están relacionados, es decir, los animales de tipo “AMP tiene el atributo TER”.

Tabla 1: Contexto Formal de Animales Vertebrados.

Fuente: [Sarmah et al., 2015].

	TER	HBLD	CBLD	FUR	PRS
AMP	$\times$		$\times$		
RPT	$\times$		$\times$		
AVS	$\times$	$\times$			
PSC		$\times$	$\times$		
CYL			$\times$		$\times$
MML	$\times$			$\times$	
MRP		$\times$	$\times$		$\times$
LPT			$\times$		$\times$

Es preciso mencionar que un contexto formal acepta tipos de conjuntos muy variados, en otras palabras, cualquier tipo de conjunto de objetos o atributos pueden ser parte del contexto formal. Así mismo, se pueden intercambiar los conjuntos de objetos y atributos de tal manera que, si (G, M, I) es un contexto formal, entonces (M, G, I^{-1}) también sería un contexto formal. Este último contexto, es llamado Contexto Dual [Ganter et al., 2016].

2.1.2. Operadores de Derivación

Una vez definido la idea de contexto formal, se pueden definir las operaciones que permitirán realizar cálculos que servirán para el descubrimiento de conceptos. Estas herramientas se utilizarán para enumerar los objetos en común que tienen los elementos del conjunto de atributos y los atributos en común que tienen algunos objetos.

Definición 2 (Operadores de Derivación) Sea (G, M, I) un contexto formal, para todo $A \subseteq G$ y $B \subseteq M$, se definen los siguientes dos operadores de derivación:

$$1. A^\uparrow = \{m \in M; \forall g \in A((g, m) \in I)\}$$

$$2. B^\downarrow = \{g \in G; \forall m \in B((g, m) \in I)\}$$

En otras palabras, el operador $A^\uparrow$ devuelve el conjunto máximo de atributos que tengan en común todos los objetos de A , y el operador $B^\downarrow$ nos entrega el conjunto máximo de objetos que tienen en común todos los atributos existentes en B [Liu et al., 2017].

Comúnmente, los operadores de derivación suelen denotarse también como A' o B' , sin distinguir entre $\uparrow$ y $\downarrow$, por lo que el sentido de la derivación radica en si el conjunto es de atributos o de objetos, siempre y cuando se cumplan las condiciones establecidas en su definición.

Adicionalmente, para A , un subconjunto de objetos, A' será un conjunto de atributos. Si se aplica la derivación a A' , es decir, $(A')'$, este devolverá un conjunto de objetos. Esta acción suele llamarse *segunda derivación* y se denota A'' (o $(A^\uparrow)^\downarrow$). También se puede realizar el mismo proceso con B , un conjunto de atributos, y su segunda derivada sería B'' (o $(B^\downarrow)^\uparrow$), que también sería un conjunto de atributos.

A partir de lo anterior, se definen las siguientes propiedades:

Propiedad 1 Para los subconjuntos $A, A_1, A_2 \subseteq G$, se tiene:

1. $A_1 \subseteq A_2 \Rightarrow A'_2 \subseteq A'_1$
2. $A \subseteq A''$
3. $A' = A'''$

Adicionalmente, para los subconjuntos $B, B_1, B_2 \subseteq M$, se tiene:

1. $B_1 \subseteq B_2 \Rightarrow B'_2 \subseteq B'_1$
2. $B \subseteq B''$
3. $B' = B'''$

Estas propiedades, provienen de la teoría llamada Conexiones de Galois, donde se demuestra y explica el origen de estas propiedades en los operadores de derivación y sus subconjuntos [Ganter y Wille, 1999b].

2.1.3. Conceptos Formales

Una vez que ya se han definido los operadores de derivación aplicables a un contexto formal, se pueden definir los conceptos formales.

Definición 3 (Concepto Formal) Dado un contexto formal (G, M, I) , un concepto formal es una tupla (A, B) tal que:

- $A \subseteq G$
- $A' = B$
- $B \subseteq M$
- $B' = A$

donde el conjunto A es llamado extensión y el conjunto B es llamado intención. De ello, resulta necesario decir, que, un concepto formal se compone de dos partes: su extensión, vale decir, los objetos que tienen en común todos los atributos de la intención. Por otro lado, la intención esta constituida precisamente por los atributos que tienen en común todos los elementos de extensión [Sarmah et al., 2015].

Dada esta definición de conceptos formales, se puede decir que un concepto formal corresponde a un sub tabla del contexto formal, donde sus filas son los elementos de la extensión (A) y las columnas la intención (B).

Ejemplo 1 Considerando lo ejemplificado en de la tabla 1, se puede observar que el la tupla $(\{CYL, MRP, LPT\}, \{CBLD, PRS\})$ es un concepto formal. Para comprobar esto, se debe aplicar los operadores de derivación para cumplir con la definición de concepto formal.

- $A \subseteq G$
- $B \subseteq M$
- Ya que $A = \{CYL, MRP, LPT\}$, entonces, $A' = \{CBLD, PRS\}$. Por lo que $A' = B$.
- Como $B = \{CBLD, PRS\}$, entonces, $B' = \{CYL, MRP, LPT\}$. Por lo que $B' = A$.

Por lo tanto, el par $(\{CYL, MRP, LPT\}, \{CBLD, PRS\})$ es un concepto formal.

2.1.4. Jerarquía de Conceptos

Posterior a la definición de los conceptos formales y dada la generalización de los contexto formales, se puede plantear un contexto formal inspirado en algunos animales (tabla 1). Intuitivamente, se puede notar que hay cierta jerarquía u orden entre los conceptos que se puedan encontrar.

Por ejemplo, en un contexto sobre animales, los seres humanos comparten algunas la características con los equinos y otros especies. Aquella categoría suele llamarse *animales mamíferos*. Se puede entender que estos mamíferos son un concepto que contiene a los humanos, equinos y el resto de los que compartan dichas características. Sin embargo, si en el mismo ejemplo se selecciona la categoría de animales ovíparos (que contienen a las aves, algunos

peces, entre otros), y se intenta ordenar con el concepto mamífero, se puede denotar que estos no son comparables. Esta característica se debe a que los conceptos formales están *parcialmente ordenados*.

Definición 4 (Orden Parcial) Sea P un conjunto, es un conjunto *parcialmente ordenado* (o *poset* en inglés) si la siguiente relación binaria no se cumple para todos para $x, y, z \in P$:

- $x \leq x$ (*Propiedad de Reflexividad*)
- $x \leq y \wedge y \leq x \Rightarrow x = y$ (*Propiedad de Antisimetría*)
- $x \leq y \wedge y \leq z \Rightarrow x \leq z$ (*Propiedad de Transitividad*)

En otras palabras, un conjunto *parcialmente ordenado*, es un conjunto donde existen elementos que cumplen con una relación de orden pero también hay otros que no, lo que los hace *incomparables* entre sí.

En el ejemplo anterior, el concepto ovíparo no es menor ni mayor que el concepto mamífero, pero tampoco son equivalentes. En otros términos, no se pueden ordenar estos dos conceptos como sí se puede hacer con los conceptos contenidos en cada una de estas categorías. Para poder encontrar este orden parcial en el conjunto de conceptos formales, se debe definir las jerarquías de conceptos.

Definición 5 Dadas las tuplas (A_1, B_1) y (A_2, B_2) , ambos conceptos formales de un contexto (G, M, I) , se puede decir que (A_1, B_1) es **subconcepto** de (A_2, B_2) , y a su vez, (A_2, B_2) es un **superconcepto** de (A_1, B_1) si se cumple lo siguiente:

$$\blacksquare (A_1, B_1) \leq (A_2, B_2) \iff A_1 \subseteq A_2$$

Vale mencionar que $A_1 \leq A_2$ es equivalente a que $B_2 \subseteq B_1$. El símbolo $\leq$ se ocupa para indicar esta relación [Ganter et al., 2016].

2.1.5. Retículos

Los retículos, también llamados *lattices*, son una estructura matemática proveniente de la teoría de conjuntos. Esta estructura es un conjunto que, junto con tener la propiedad de ser *parcialmente ordenado*, cumple con la característica de que para cualquier par de elementos del conjunto, existe un único supremo y un único ínfimo. Para dicho par de elementos, el supremo será la mínima cota superior entre ambos, en otras palabras, dichos elementos del conjunto tendrán un único elemento superior en común. Por otro lado, el

ínfimo será la máxima cota inferior entre dicho par, vale decir, ambos elementos tendrán un único valor inferior en común [González *et al.*, 2003].

Desde el punto de vista del álgebra, para cualquier elemento a, b, c , elementos del retículo, se cumplen las siguientes propiedades:

- $a \wedge b = b \wedge a$ (Ley de Conmutatividad)
- $a \wedge (b \wedge c) = (a \wedge b) \wedge c$ (ley de Asociatividad)
- $a \wedge (b \vee a) = a$ (Ley de absorción)
- $a \wedge a = a$ (Ley de Idempotencia)

Esta propiedades se cumplen tanto para el operador $\wedge$ como para $\vee$.

Definición 6 El conjunto de todos los conceptos formales de un contexto formal (G, M, I) , con el orden de sus superconceptos y subconceptos, cumpliendo además con las propiedades de un retículo, es llamado **Lattice Conceptual** del contexto formal. La lattice conceptual suele denotarse como $\mathfrak{B}(M)$ [Ganter *et al.*, 2016].

Este *lattice conceptual* puede ser representado mediante un diagrama de Hasse, que busca mostrar una versión simplificada del un conjunto parcialmente ordenado. En la tabla 2, se puede observar un contexto formal sobre números divisores. G está compuesto por los números $\{2, 3, 6, 9, 12, 18\}$, M es igual a $\{2, 3, 6, 9, 12, 18\}$, e I es la relación binaria que verifica que para algún $g \in G$ y $m \in M$, g sea divisor de m .

Tabla 2: Contexto Formal números divisores
Fuente: Elaboración propia.

	2	3	6	9	12	18
2	×		×		×	×
3		×	×	×	×	×
6			×		×	×
9				×		×
12					×	
18						×

En este contexto, mediante el uso de los operadores de derivación y la definición de concepto formal, se pueden encontrar los siguientes conceptos formales:

- $(\emptyset, \{2,3,6,9,12,18\})$
- $(\{2,3,6,9,18\}, \{18\})$
- $(\{2,3,6,12\}, \{12\})$
- $(\{2,3,6\}, \{6,12,18\})$
- $(\{3,9\}, \{9,18\})$
- $(\{2\}, \{2,6,12,18\})$
- $(\{3\}, \{3,6,9,12,18\})$
- $(\{2,3,6,9,12,18\}, \emptyset)$

Dado estos conceptos formales, se debe obtener el orden parcial entre los conceptos para construir una *lattice*. Luego de esto, la *lattice conceptual* resultante puede ser representada en un diagrama de Hasse como el presentado en la figura 2.

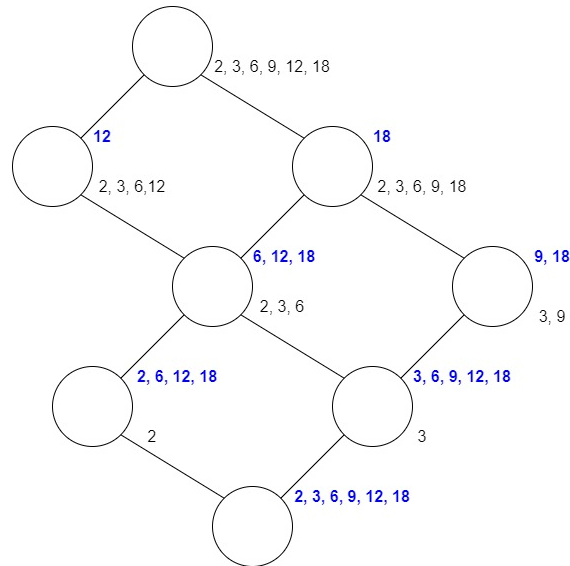


Figura 2: Diagrama de Hasse
Fuente: Elaboración propia (a partir de resultado en LatViz).

En el diagrama de Hasse de la figura 2, se puede identificar el orden parcial dado por los superconceptos y subconceptos, y las propiedades de una *lattice*. Por ejemplo, entre los conceptos $(\{2, 3, 6\}, \{6, 12, 18\})$ y $(\{3, 9\}, \{9, 16\})$, se puede notar que entre ambos existe un solo superconcepto en común o supremo que es el concepto $(\{2, 3, 6, 9, 18\}, \{18\})$. A su vez, se puede observar el único ínfimo o subconcepto en común que es $(\{3\}, \{3, 6, 9, 12, 18\})$.

2.2. Clausuras

Las clausuras y los sistemas de clausuras son conceptos un tanto abstractos, pero de fácil definición. En un principio, se puede mencionar que los sistemas de clausuras son conjuntos de conjuntos que cumplen ciertas propiedades (las propiedades definidas para la clausura).

Al trabajar con conjuntos de conjuntos, se utilizarán las letras mayúsculas para conjuntos y las letras minúsculas para elementos de dichos conjuntos. Para poder obtener información sobre cualquier tipo de conjunto de conjuntos, primero se debe definir las siguientes dos operaciones. Sea S un conjunto de conjuntos, tenemos [Ganter *et al.*, 2016]:

- Para obtener los elementos que existen en total, se deben **unir** todos los conjuntos contenidos en S , es decir:

- $\cup_S := \{x | \exists_{S \in S} x \in S\}$

- Para obtener los elementos que ocurren entre todos los conjuntos, se debe **interseccionar** cada uno de los conjuntos de S , es decir:

- $\cap_S := \{x | \forall_{S \in S} x \in S\}$

Esta definición, funciona para casi todos los tipos de conjuntos. Excepcionalmente, para $S = \emptyset$, se define $\cup_{\emptyset} := \emptyset$ y $\cap_{\emptyset} := M$. Este caso, se puede entender porque en el ejemplo resuelto del contexto 2, el concepto formal (1) estaba formado por el conjunto vacío y todo el conjunto de atributos, vale decir, la tupla $(\emptyset, M)$.

2.2.1. Definición

Un sistema de clausuras sobre un conjunto M de atributos, es un conjunto compuesto por subconjuntos que contienen M , y que es cerrado en sus intersecciones.

Definición 7 (Sistema de Clausuras) *Dado un conjunto M y un conjunto $C \subseteq \mathfrak{B}(M)$, para que exista un sistema de clausuras sobre M , se debe cumplir:*

1. $M \in C$
2. Sea $D \subseteq C$, entonces $\cap_D \in C$

*En esta definición, se puede observar que la primera condición está contenida en la segunda, excepto para el caso que $D = \emptyset$, debido a lo explicado anteriormente [Ganter *et al.*, 2016].*

2.2.2. Operadores de Clausura

Los operadores de clausura son aquellos que permiten construir los sistemas de clausuras sobre un conjunto, verificando que se cumplan las reglas del operador de clausura. Estos operadores de clausura pueden ser variados y adaptados al tipo o características de conjuntos sobre los que se está trabajando. Sin embargo, estos operadores deben cumplir con las siguientes condiciones para poder ser utilizados:

Definición 8 (Operador de Clausura) Sea un operador de Clausura φ sobre un conjunto M , este puede ser utilizado para asignar clausuras a $\varphi X \subseteq M$, siendo X cualquier posible subconjunto de M , este debe cumplir que:

- $X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$ (Cumplir con la Monotonía)
- $X \subseteq \varphi(X)$ (Cumplir con la Extensividad)
- $\varphi(\varphi(X)) = \varphi(X)$ (Cumplir con la Idempotencia)

2.3. Programación Paralela

Con el rápido desarrollo de las tecnologías computacionales de hoy en día, la computación paralela a tenido que evolucionar durante las últimas décadas en términos de diseño, modelos y lenguajes de programación para aprovechar esta ventaja tecnológica.

La computación paralela trata sobre dividir una tarea en subtareas, independientes entre si. Estas subtareas, que pueden ser de igual o diferente tamaño, son asignadas a procesos o hebras durante el inicio de la resolución de la tarea o durante el transcurso de su ejecución. Estas hebras necesitan ser coordinadas para su correcta ejecución, para eso, debe existir un proceso de asignación de tareas, envío variables de entrada y de retorno de resultados. Este proceso de orquestación de las tareas es llamado *Scheduling*. El periodo de tiempo requerido para la creación de los procesos que ejecutan las subtareas, sumado al tiempo que se ocupa en el *Scheduling* es llamado *Overhead*. Resumiendo en otras palabras, el *Overhead* de una tarea paralela es la suma de los tiempos que hacen relación a la organización, administración y orquestación que permiten realizar dicha tarea de forma paralela y no tiene relación con el problema a resolver en las subtareas. De esta definición se puede notar que aunque esta forma de programación nos da el beneficio de avanzar de manera simultanea en diferentes partes de nuestra tarea a resolver, hay que considerar el costo del *Overhead* al momento del diseño del algoritmo.

Es el proceso en el que las tareas son asignadas a los procesos o hilos, y se les da un orden de ejecución. Este puede ser especificado en el código, en tiempo de compilación o dinámicamente en tiempo de ejecución. El proceso de scheduling debe tener en cuenta la dependencia entre tareas, ya que, aunque muchas pueden ser independientes, otras pueden requerir los datos producidos por otras tareas.

2.3.1. Patrones de Programación Paralela

En [Li y Zhang, 2018] se presentan algunos patrones de diseño en algoritmos paralelos o programas que funcionan en paralelo.

- **Patrón Maestro/Esclavo:** Este patrón consiste en dos partes: Maestro/Esclavo. El Maestro es quien se encarga de la repartición de tareas o la recolección de los resultados de cada subtarea para la obtención de la solución final del trabajo realizado. Por otro lado, los esclavos son quienes las tareas de forma recurrente, en otras palabras, ejecutan las subtareas, reciben las señales o instrucciones y devuelven los resultados al maestro.

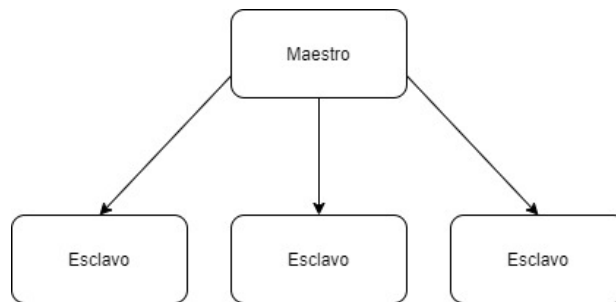


Figura 3: Patrón Maestro/Esclavo
Fuente: Elaboración propia.

- **Patrón Dividir y conquistar:** Esta estrategia algorítmica se refiere a la división de la tarea en subproblemas que puedan resolverse de manera independiente y que solamente requiere una recopilación final de los resultados para la obtención de la solución final.

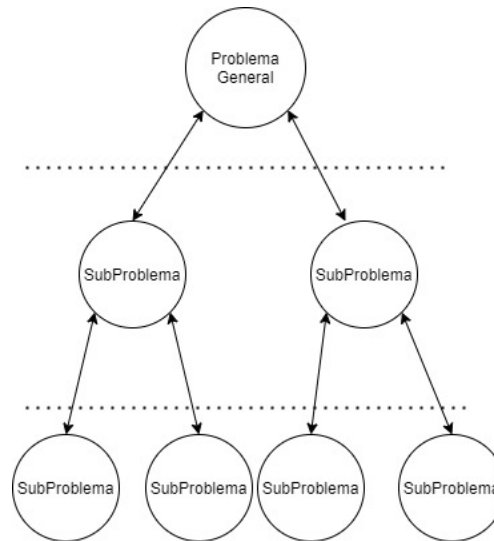


Figura 4: Patrón Dividir y Conquistar
Fuente: Elaboración propia.

- **Patrón Pipeline:** Este patrón consiste en la ejecución de subtareas del problema a resolver en diferentes etapas o tiempos. A diferencia de lo anterior, este patrón de paralelización hace alusión a la ejecución de los subprocesos al momento que sea posible realizar la subtaska mediante una señal que indique que los requisitos para su inicio están listos.

2.3.2. Entornos de ejecución para la programación paralela

En [Li y Zhang, 2018] también se detallan algunos de los posibles entornos de ejecución donde hoy en día se suelen ejecutar tareas utilizando la programación paralela.

- **Computación en la nube:** La computación en la nube o computación distribuida, se basa en el uso de la computación paralela. Esta consiste en la demanda de decenas de miles de solicitudes por parte de aplicaciones o usuarios, que se comunican a través de la internet. La capacidad de escalabilidad y el modelo de negocios para la adquisición de recursos en la nube, a permitido al auge de los sistemas y tecnologías distribuidas que se basan en los patrones de diseño de la programación paralela.
- **Arquitectura de varios núcleos:** En los últimos años se ha evidenciado el avance de la integración de procesadores con una mayor capacidad de cómputo y un aumento en la cantidad de núcleos. Esta arquitectura de múltiples núcleos, hace uso de los recursos del mismo computador de manera efectiva buscando explotar la capacidad de ejecutar diferentes procesos en paralelo.

- **Computadores personales de alto rendimiento:** Este último entorno hace alusión al uso de computadores que poseen mejor rendimiento que estaciones de trabajo ordinarias. Estos computadores personales permiten manejar de mejor manera un mayor número de datos y la realización de intercambio de mensajes y sincronización.

2.3.3. Balance de Carga

Para obtener un buen rendimiento, de los algoritmos paralelos requieren que los procesos que tiene en ejecución tengan una duración en tiempo similar o equitativa, debido a que si un proceso tarda mucho más que el resto, el tiempo que tarda el algoritmo en resolver la tarea pasa a depender cuanto tarda este proceso más extenso por lo que se desaprovecha el uso de la paralelización. Esto ocurre cuando hay una repartición desigual entre el tamaño de las subtareas que deben hacer los diferentes procesos. Esta problemática se conoce como balance de carga.

Para solucionar este problema, en [Naiouf y De Giusti, 2002] se explican dos posibles enfoques para resolver la tarea del balance de carga:

- **Balance de carga estático:** Este enfoque habla de una distribución de trabajo que se define en la fase de inicio del algoritmo. La repartición de trabajo definida por un parámetro al comienzo de la ejecución genera malos resultados si se esta trabajando en problemas donde la carga de trabajo es variable a lo largo de la ejecución.
- **Balance de carga dinámico:** En esta estrategia se busca resolver la tarea de repartición de trabajo de acuerdo a la información del momento en el que se esta ejecutando el proceso, ocupando intercambio de mensajes entre procesos respecto a su estado. Esta estrategia, requiere mantener una visión global del sistema e implementar las condiciones necesarias para la repartición de tareas durante la ejecución, como por ejemplo un proceso que tome las decisiones, recopile la información y asigne subtareas.

2.4. Tipos de Enumeraciones

La enumeración es una operación sobre conjuntos que permite obtener un listado de los elementos pertenecientes a un conjunto. Para obtener todos los conceptos formales de un contexto formal en FCA, se necesita recorrer todo el espacio de búsqueda, que es equivalente a enumerar del conjunto potencia de los atributos. Es por esto que definir una forma de enumeración que nos permita obtener ventajas algorítmicas en el proceso de búsqueda de los conceptos normales es de suma importancia. A continuación, se presentan algunas de las formas de enumeración existentes.

2.4.1. Enumeración Lexicográfica

El orden lexicográfico, es un orden completo en una colección de conjuntos. Este enumeración suele denominarse este orden diccionario, ya que las reglas de la enumeración son las mismas que las que se utilizan para ordenar las palabras en un diccionario. A partir del orden lineal entre los elementos, se enumera por el orden del producto cartesiano entre ellos, de esta manera, dada una tupla (a_1, b_1) y (a_2, b_2) , $(a_1, b_1) \leq (a_2, b_2)$ solo si $a_1 \leq a_2$ o $a_1 = a_2$ y $b_1 \leq b_2$.

En general, sean los elementos $a = (a_1, a_2, \dots, a_i, \dots, a_{n_a})$ y $b = (b_1, b_2, \dots, b_i, \dots, b_{n_b})$ de largo n_a y n_b respectivamente, decimos que a es lexicográficamente menor a b si y sólo si [Ganter y Wille, 1999b]:

$$\exists_{j \in [1..n_a]} (a_j < b_j \text{ y } \forall_{i < j} (a_i = b_i))$$

Ejemplo 2 Sea el conjunto $A = \{a, b, c\}$, se busca recorrer dicho conjunto, y sus posibles combinaciones, utilizando la enumeración lexicográfica. Asumiendo que existe la relación de orden $a < b < c$, la enumeración léctica sería:

1. $\{a\}$
 - $\{a, b\}$
 - $\{a, b, c\}$
 - $\{a, c\}$
2. $\{b\}$
 - $\{b, c\}$
3. $\{c\}$

2.4.2. Enumeración Léctica

El orden léctico es un orden completo en una colección de conjuntos. Particularmente, se utiliza como orden en el conjunto potencia del conjunto de atributos M . En este, se asume que existe un orden lineal entre los elementos, y se busca enumerarlos de menor a mayor. Esto último incluye las combinaciones entre los elementos del conjunto, vale decir, entre la enumeración de dos elementos del conjunto, se recorrerán todas las posibles combinaciones que contengan elementos menores en el orden determinado entre los elementos del conjunto.

A es léxicamente menor a B si y sólo si [Ganter *et al.*, 2016]:

$$A <_m B \iff (m \in B, m \notin A, \forall_{n <_m} (n \in A \iff n \in B))$$

En otras palabras:

$A <_m B$ si y solo si m es el elemento más pequeño en el que A y B difieren y $m \in B$.

Ejemplo 3 Sea el conjunto $A = \{a, b, c\}$, se busca recorrer dicho conjunto, y sus posibles combinaciones, utilizando la enumeración léctica. Asumiendo que existe la relación de orden $a < b < c$, la enumeración léctica sería:

1. $\{c\}$
2. $\{b\}$
 - $\{b, c\}$
3. $\{a\}$
 - $\{a, c\}$
 - $\{a, b\}$
 - $\{a, b, c\}$

Dado el ejemplo anterior, el orden de la enumeración de los elementos del conjunto A comienza por el orden lineal existente entre los elementos del conjunto. El orden lineal utilizado en este caso, es el orden alfabético invertido. De esta manera, la enumeración queda de la siguiente manera: c, b, cb, a, ac, ab, abc . Cabe mencionar, que en este ejemplo no se incluyó el conjunto vacío para no extender la demostración pero debe considerarse al utilizar este tipo de enumeración.

2.5. Algoritmos Secuenciales

En esta sección, se detallarán algunos de los algoritmos secuenciales más importantes utilizados en el Análisis Formal de Conceptos. Algunos de ellos también son ocupados en otras áreas de estudio como *Data Mining* o *Minería de Datos*.

2.5.1. Algoritmo NextClosure

Uno de los algoritmos más utilizados en la rama de Análisis Formal de Conceptos es Next-Closure. Dicho algoritmo, se encarga de buscar todas las clausuras del contexto formal. Para

esta tarea, NextClosure ocupa la enumeración léctica del conjunto de objetos que le permite optimizar la forma de recorrer el espacio de búsqueda [Bazin y Ganascia, 2016]. Por lo anterior, se puede decir que el enfoque o naturaleza de NextClosure es de *depth-first* o hacer una búsqueda en profundidad.

El algoritmo NextClosure realiza una búsqueda de conceptos formales recorriendo el espacio de búsqueda utilizando la enumeración léctica, vale decir, comienza desde el conjunto vacío y termina con la combinación de todos los elementos del conjunto [Bazin y Ganascia, 2016].

Para cada iteración, NextClosure calcula la clausura para la combinación que se está revisando, ya sea un elemento del conjunto o una mezcla entre ellos. Si dicha combinación no cumple con las propiedades para ser un concepto formal, este avanzará sin calcular clausuras hasta llegar a la clausura antes determinada o hasta que el menor mayor elemento de la combinación, por el orden dado en la enumeración léctica, cambie. Esta propiedad que permite evitar calcular clausuras de combinaciones se llama test canónico. Esta propiedad nos permite asegurar que, si para una combinación dada su clausura está más adelante en la enumeración, podemos evitar calcular las siguientes clausuras hasta llegar a la combinación calculada, siempre y cuando dicha clausura este dentro de la misma ramificación de la enumeración léctica.

A continuación se presenta un pseudocódigo del funcionamiento del Algoritmo NextClosure:

Algorithm 1 Calculo Conceptos con NextClosure

Input:

T = Conjunto de objetos
I = Conjunto de atributos

Output: C = Lista de Candidatos

R = Lista vacía
P = CalcularClausura({},T,I)
while P != I **do**
 R.agregar(P)
 P = NextClosure(P,T,I)
end while
return R

Algorithm 2 NextClosure

Input:

T = Conjunto de objetos
I = Conjunto de atributos
P = Combinación actual en la enumeración léctica

Output: Paux = Concepto formal encontrado o I en caso de no encontrar ninguno

```
for i en I ordenado alfabéticamente invertido do
  if Letra i está en P then
    quitar letra i de P
  else
    Paux = CalcularClausura(P.union(i),T,I)
    if i < Mínimo(Paux-P) then
      return Paux //Ya que es un concepto formal
    end if
  end if
end for
return I
```

2.5.2. Algoritmo Apriori

El algoritmo Apriori fue uno de los primeros algoritmos desarrollados para encontrar reglas de asociación en minería de datos [Agrawal y Srikant, 1994]. Este algoritmo busca recorrer o enumerar todas las posibles combinaciones del conjunto de elementos utilizando el concepto de *breadth-first*, vale decir, este algoritmo va recorriendo las posibles combinaciones de elementos por *niveles*, partiendo desde los elementos del conjunto, luego todas las posibles combinaciones que contengan solo 2 elementos, luego todas las combinaciones de a 3 elementos, así sucesivamente hasta llegar a una combinación que contenga todos los elementos del conjunto.

El algoritmo Apriori utiliza el concepto de *candidatos*, que son todas las clausuras de combinaciones que no fueron cerradas en un nivel. Estos candidatos serán los puntos de partida para analizar las combinaciones que contengan dichos elementos en un próximo nivel.

A continuación, se presenta un pseudocódigo del algoritmo Apriori:

Algorithm 3 GenerarCandidatos

Input:

L = Datos niveles
k = Nivel actual
T = Conjunto de objetos
I = Conjunto de atributos

Output: R = Lista de Candidatos siguiente nivel

```
Lk = L.get(nivel)
for candidato en Lk do
    for letra en I do
        Paux = CalcularClausura(P.union(letra),T,I)
        L[Largo(Paux)].agregar(Paux) //Se agregar el candidato al nivel equivalente a su
largo
    end for
end for
if Existen datos en el nivel k en L then
    return L.get(k)
else
    return {}
end if
```

Algorithm 4 Apriori

Input:

T = Conjunto de objetos
I = Conjunto de atributos

Output: R = Lista de Conceptos formales

```
L = Lista vacía
for Letra i en I do
    Paux = CalcularClausura(i,T,I)
    L[Largo(Paux)].agregar(Paux)
end for
i = 0
while i<Largo(I) do
    Paux = GenerarCandidatos(L,i+1,T,I)
end while
return {}
```

2.6. Algoritmos Paralelos o Distribuidos

2.6.1. Paralelización del Algoritmo NextClosure

Dada la naturaleza de búsqueda en profundidad de NextClosure, no es trivial pensar en una paralelización de este algoritmo. La opción de aprovechar la existencia del test canónico para el evitar calcular clausuras es un objetivo latente al momento de plantear una paralelización de NextClosure.

En [de Moraes *et al.*, 2016] se plantea una paralelización del algoritmo NextClosure para calcular el conjunto mínimo de reglas de implicación de un contexto formal. El objetivo de calcular este conjunto de reglas de implicación es que a partir de este conjunto se pueden generar todas los conceptos formales de un contexto formal. En la propuesta presentada por los autores, se plantea un algoritmo del patrón de paralelización Pipeline, que calcula las reglas de implicación a diferentes subconjuntos de manera creciente, vale decir, en cada iteración del algoritmo, diferentes procesos ejecutan una adaptación del algoritmo NextClosure para el calculo de reglas de implicación en diferentes subconjuntos. Los autores plantean que uno de los problemas de la propuesta es el equilibrio del trabajo entre los diferentes procesos de NextClosure. Cabe destacar que en este algoritmo no queda claro el objetivo de la paralelización dado que generar diferentes procesos ejecutando el algoritmo NextClosure sobre el mismo conjunto podría generar redundancia, por lo que se estaría recibiendo un real beneficio de esta forma de paralelización de trabajo.

Por otro lado, en [Krajca *et al.*, 2010] se plantea un algoritmo un algoritmo paralelo recursivo que también posee un enfoque *depth-first*. En la propuesta de los autores, se presenta un algoritmo secuencial denominado GenerateFrom que lo que hace es recibir un concepto formal o un conjunto vacío y un atributo de partida, y busca el próximo concepto formal más cercano agregando atributos con un enfoque de búsqueda en profundidad. Posterior a la presentación de este algoritmo, se plantea su paralelización llamada ParallelGenerateFrom, en la que, de manera recursiva, van generando subprocesos que ejecuten GenerateFrom pero que comiencen a buscar desde atributos diferentes, logrando la paralelización de dicho algoritmo. En [Langdon *et al.*, 2011] se presenta una implementación del algoritmo ParallelGenerateFrom en CUDA comparándola con los resultados presentes en [Krajca *et al.*, 2010]. En esta comparación, la implementación en CUDA no logra obtener mejores resultados por motivos que los autores no pueden explicar de manera certera y busca argumentar a los costos asociados a los intercambios de datos entre la CPU y la GPU.

2.6.2. Paralelización del Algoritmo Apriori

El algoritmo de Apriori Paralelo es una extensión del algoritmo Apriori ya mencionado, pero paralelizando el trabajo que se realiza en cada nivel. Dada su naturaleza *breadth-first*, es intuitiva su paralelización, siendo fácil el concluir una paralelización por candidatos de cada

nivel. Sin embargo, diversos autores han generado propuestas más robustas que esto.

A continuación se presenta un pseudocódigo de una paralelización del algoritmo Apriori:

Algorithm 5 GenerarCandidatos Paralelo

Input:

L = Datos niveles
k = Nivel actual
T = Conjunto de objetos
I = Conjunto de atributos

Output: R = Lista de Candidatos siguiente nivel

```
candidatos = L.get(nivel)
Paux = CalcularClausuraParalelo(candidatos,T,I)
for resultado en Paux do
    L[Largo(resultado)].agregar(resultado)
end for
if Existen datos en el nivel k en L then
    return L.get(k)
else
    return {}
end if
```

Algorithm 6 Apriori Paralelo

Input:

T = Conjunto de objetos
I = Conjunto de atributos

Output: R = Lista de Conceptos formales

```
L = Lista vacía
for Letra i en I do
    Paux = CalcularClausura(i,T,I)
    L[Largo(Paux)].agregar(Paux)
end for
i = 0
while i<Largo(I) do
    Paux = GenerarCandidatos Paralelo(L,i+1,T,I)
end while
return {}
```

Dado que el origen del algoritmo Apriori es en la aplicación de la Minería de Datos, es

recurrente su uso en esta rama en el estudio de patrones frecuentes. Dicha tarea se vuelve compleja dado el problema que conlleva el gran número de datos con los que se trabaja hoy en día. Este problema de Minar en Big Data a llevado a algunos autores a utilizar tecnologías de MapReduce para la minería de patrones frecuentes en Big Data.

Los autores de [Luna et al., 2017], proponen un algoritmo que cumpla con lo anterior basado en Apriori utilizando MapReduce y Hadoop. Su algoritmo inicia separando los datos en diferentes grupos agrupados por atributos en común. Luego de esto, reducen los datos existentes agregando un contador de cuantas veces se repite dicha combinación de atributos. Posterior a esto, ejecutan una versión adaptada de Apriori al contexto de trabajar con multiples nodos que poseen este gran volumen de dato, a partir del resultado de MapReduce, calculan las clausuras aprovechando la reducción de datos evitando revisar cada uno de los registros.

2.6.3. Paralelización de la Enumeración Léctica

Una estrategia para paralelizar el recorrido del espacio de búsqueda de un contexto formal es a través de la descomposición de la enumeración léctica en particiones independientes. De esta forma, se define una *carga* como un subconjunto de n atributos, siguiendo la misma relación de orden lineal asignada a los elementos del conjunto en la enumeración léctica. El tamaño de la carga debe ser definido previo a la ejecución del algoritmo. Sobre los elementos de la carga se calcularán las clausuras utilizando la enumeración léctica.

Ejemplo 4 Sea el conjunto de elementos $A = \{a, b, c, d\}$, se busca enumerar el conjunto, y sus posibles combinaciones, utilizando la enumeración Paraléctica. Asumiendo que existe la relación de orden lineal $a < b < c < d$, se define la carga de largo 2 como $m_{carga} = \{c, d\}$. Se procede a realizar la enumeración léctica, obteniendo como resultado:

- $\{d\}$
- $\{c\}$
 - $\{c, d\}$

Luego de esto, se procede a hacer la iteración por niveles, combinando los elementos del nivel con la enumeración de la carga:

Nivel 1. $\{\{a\}, \{b\}\}$

- $\{a\}$

- $\{a, d\}$
- $\{a, c\}$
 - $\{a, c, d\}$
- $\{b\}$
 - $\{b, d\}$
 - $\{b, c\}$
 - $\{b, c, d\}$

Nivel 2. $\{a, b\}$

- $\{a, b\}$
 - $\{a, b, d\}$
 - $\{a, b, c\}$
 - $\{a, b, c, d\}$

Como en este caso, la carga fue definida como dos y solo quedaron dos elementos restantes, se ejecutan solo dos niveles de la enumeración paralela.

Esta estrategia, nombrada Primera Propuesta Paralelización Léctica (PPLP), busca optimizar el desempeño del calculo de clausuras en un computador aprovechando y utilizando los múltiples núcleos de este.

Específicamente, esta estrategia inicia definiendo la carga que se busca paralelizar. Luego de tener la carga definida y realizada la búsqueda de los conceptos formales sobre la esta, el algoritmo itera en *niveles*, al igual que el Algoritmo Apriori, buscando candidatos y siguiendo el mismo principio que dicho algoritmo. Estos niveles trabajan sobre el resto de los elementos que no están presentes en la carga, de esta manera, en cada nivel se hará una mezcla entre las combinaciones del nivel que se está revisando y la carga.

A continuación, se presenta un pseudocódigo de esta estrategia:

Algorithm 7 Estrategia de Paralelización Léctica

Input:

T = Conjunto de objetos
I = Conjunto de atributos
TamCarga = Conjunto de atributos

Output: R = Lista de Conceptos formales

```
R = Lista vacía
Candidatos = Lista vacía
Carga = DefinirCarga(TamCarga,T,I)
Paux = CalcularClausuras(Carga,T,I)
R.add(Paux)
Candidatos.agregar(Paux)
for i en rango(1,Largo(I)-TamCarga) do
    Paux=DistribuirTrabajo(Candidatos.get(i),NextClosure,T,I)
    //Distribuye los candidatos del nivel i entre varios procesos que ejecutan NextClosure
    R.add(Paux)
    Caux = GenerarCandidatos(Candidatos,i,T,I)
    Candidatos[i+1].agregar(Caux)
end for
return {}
```

La estrategia anterior administra una carga de trabajo, aunque no necesariamente esta tiene consideraciones sobre el balance de la misma. Esto, pues el espacio de búsqueda de clausuras no necesariamente es uniforme lo cual redundo en que muchos procesos puedan quedar *idle* mientras gran parte del cómputo se divide entre pocos procesos.

Una segunda estrategia, nombrada Segunda Propuesta Paralelización Léctica (SPPL), es diseñada para administrar el balance de carga de mejor manera propone avanzar en un enfoque *depth-first*, de manera que varios subprocesos trabajan en paralelo en subconjuntos del contexto formal. En esta caso, la cantidad de hebras va aumentando si un proceso tarda más que un tiempo predeterminado. Cuando un proceso supera un umbral de tiempo, se divide el espacio de búsqueda sobre el cual esta trabajando y se generan un nuevo proceso al cual se le asigna la mitad de este. Esto sucede progresivamente de manera que cuando más tarda una ejecución de este proceso, más subprocesos se generan.

A continuación, se presenta una imagen que ejemplifica como se va dividiendo una ejecución de un contexto formal compuesto por cinco atributos $\{a, b, c, d, e\}$:

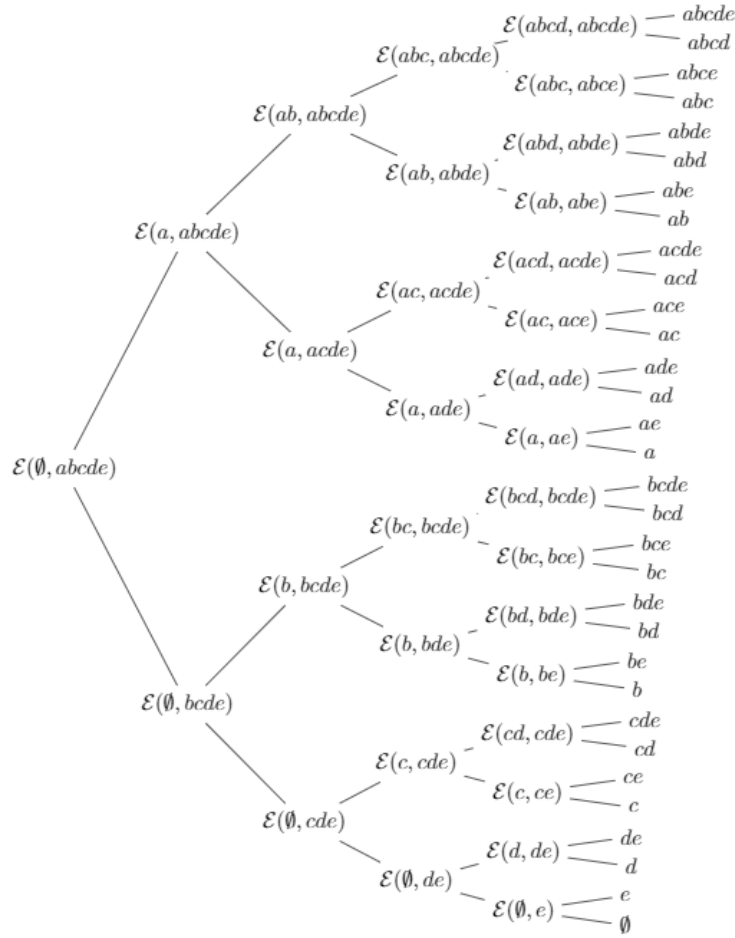


Figura 5: Ejemplo división de trabajo SPPL
Fuente: Profesor Victor Codocedo.

2.6.4. Otros Algoritmos

El algoritmo ParaPruning es una propuesta presentada en [Ma *et al.*, 2010]. Este algoritmo propone factorizar el contexto formal de manera que se puedan calcular, de manera paralela, las clausuras en cada uno de los sub-contextos formales resultantes de la factorización. Sin embargo, en la demostración del mismo algoritmo, se puede notar que dicha factorización puede no ser óptima desde el punto de vista computacional ya que dicha factorización puede ser de tamaños completamente diferente, implicando que la carga de trabajo para los diferentes procesos sea desbalanceada.

En [Xiao y Hu, 2019] también se presenta una propuesta que las tecnologías de MapReduce y Hadoop, en este caso para la búsqueda de patrones frecuentes. En esta, el algoritmo llamado MRclose se basa en la idea de particionar el espacio de búsqueda, utilizando un árbol de

prefijos de patrones frecuentes o *FP-tree*, paralelizar las búsquedas en profundidad. MRClose utiliza HDFS de Hadoop para el almacenamiento del gran volumen de datos, y MapReduce para generar una lista de posibles reglas de asociación. Posterior a esto, MRClose trabaja en paralelo utilizando un enfoque en profundidad para recorrer la *lattice*. Finalmente, los resultados de los procesos son recolectados por MapReduce para pasar por un proceso final de verificación de las reglas de asociación encontradas a nivel global del algoritmo.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

En este capítulo se abarcará el diseño e implementación de la librería de los algoritmos de FCA, donde se detallarán los algoritmos seleccionados para su implementación, junto con los *datasets* que se utilizarán como prueba y los medios que se ocuparán para estas comparaciones.

3.1. Algoritmos Seleccionados

En base a la investigación realizada y a los algoritmos encontrados, se decidió realizar la implementación de los algoritmos secuenciales Apriori y NextClosure. Se escogen estos dos algoritmos dado que son los más citados o reconocidos algoritmos secuenciales en base a lo investigado. Dado la importancia que tiene el algoritmo Apriori en el mundo de minería de datos, sobre todo en reglas de asociación, y por otro lado, la relevancia de los buenos resultados de NextClosure por el uso del test canónico, los convierte en buenos candidatos para este trabajo.

Además de estos, se decidió implementar Apriori Paralelo, para poder encontrar evidencias de los beneficios de la paralelización. Junto con esto, se optó por implementar la propuesta PPPL, que buscan integrar los enfoques de Apriori y NextClosure, y segunda versión SPPL que se preocupa del balance de carga de trabajo.

A continuación se presenta una tabla de los algoritmos secuenciales elegidos para este estudio:

Tabla 3: Algoritmos Secuenciales Seleccionados
Fuente: Elaboración propia.

Nombre	Tipo de Búsqueda
Apriori	Breadth-First
NextClosure	Depth-first

A continuación se presenta una tabla de los algoritmos paralelos elegidos para este estudio:

Tabla 4: Algoritmos Paralelos Seleccionados

Fuente: Elaboración propia.

Nombre	Funcionamiento	Patrón Paralelo
Apriori Paralelo	Breadth-First	Maestro/Esclavo
PPPL	Breadth-First/Depth-first	Maestro/Esclavo
SPPL	Depth-first	Dividir y conquistar

3.2. Datasets Seleccionados

Se seleccionaron por siete *datasets* de diferente tamaño, número de atributos, número de objetos y número de conceptos formales existentes.

A continuación se presenta una tabla con las características de los *datasets* seleccionados:

Tabla 5: Datasets Seleccionados

Nombre	Número de objetos	Número de atributos	Número de Conceptos Formales
Adult	388	14	13357
Car	1217900	7	128
Credit	213116	15	14560
Flight	3543	108	241239
Horse	44850	28	939541
Mushroom	1663	22	930841
Servo	5	7240	32

3.3. Diseño Librería

Debido a que el contexto en el que se desenvuelven los algoritmos que se presentaron en este documento es de ciencia de datos, es altamente deseable utilizar la metodología de trabajo CRISP-DM.

Esta metodología se caracteriza por permitir asociar diferentes proyectos o funcionamientos en un mismo resultado. CRISP-DM se compone de 6 etapas: Comprensión del Negocio o Contexto, Comprensión de Datos, Preparación de los Datos, Modelo, Evaluación y Despliegue.

A continuación, se detalla como se aplican estas etapas a trabajo de esta memoria:

- **Comprensión del Negocio o Contexto:** Este punto de la metodología se asocia a como funcionan los algoritmos que se seleccionaron a implementar, que datos utilizan y condiciones necesitan para su funcionamiento.

- **Comprensión y Preparación de los datos:** Estas etapas se relacionan a comprender como adaptar los datos de los *datasets* que se ocuparán como contextos formales para este estudio.
- **Modelado e Implementación:** En esta etapa, se trabaja sobre la implementación de los diferentes algoritmos seleccionados.
- **Evaluación y Despliegue:** Estas últimas etapas se refieren a probar el correcto funcionamiento de la librería y la corrección de errores para su utilización en para obtener los datos de estudio.

3.3.1. Utilidades de la Librería

A continuación se presenta una lista de las funcionalidades que debe se desea obtener para el estudio:

- **Recibir Diferentes Datasets:** La librería debe ser capaz de poder recibir los diferentes *datasets* que se utilizarán para este estudio para permitir la ejecución de los diferentes algoritmos.
- **Implementación de Algoritmos Secuenciales:** En base a lo revisado en el marco conceptual de este trabajo, la librería debe contener y permitir ocupar los algoritmos secuenciales seleccionados para el estudio de presentado en este documento.
- **Implementación de Algoritmos Paralelos:** En base a los algoritmos seleccionados, se debe estudiar y desarrollar la paralelización de Apriori, junto con la implementación de las dos versiones de la propuesta del algoritmo Paraléctico.
- **Guardar Resultados Desempeño Algoritmo:** La librería debe ser capaz de poder guardar en un archivo los resultados del desempeño de los algoritmos ejecutados bajo un *dataset* determinado.

3.4. Entorno de Ejecución

Los algoritmos fueron ejecutados en un servidor Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, con 48 núcleos, con 126 GB RAM, con sistema operativo Ubuntu Server 18.04.5 LTS. Para la paralelización de los algoritmos se utilizó la librería Ray para Python 3.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En el presente capítulo se muestran las implementaciones de los algoritmos y de la librería junto con los resultados obtenidos de las ejecuciones de los algoritmos en los diferentes datasets seleccionados. Estos resultados se presentan en tablas y en diferentes gráficas a lo largo de este capítulo.

4.1. Implementación de Librería

Como resultado de la implementación de la librería utilizando la metodología CRISP-DM, se desarrolló la librería resultante que es utilizada en el estudio de esta memoria. Esta se encuentra publicada en el siguiente enlace. Esta librería contiene la implementación de los algoritmos Apriori, Apriori Paralelo, NextClosure, PPPL y SPPL. Esta librería soporta la ejecución de todos los *datasets* seleccionados para este estudio.

Para instalar la librería se debe seguir las instrucciones que aparecen en el repositorio en Github que la contiene. Para hacer uso de la librería, se debe utilizar la función *Algorithm_FCA*. Esta función recibe tres parámetros: *Algorithm*, *Dataset*, *Carga*.

- **Algorithm:** Corresponde al nombre del algoritmo que se desea utilizar. Los posibles nombres a escribir son: Apriori, Apriori_Parallel, NextClosure, PPPL, SPPL. Este parámetro es obligatorio.
- **Dataset:** Esta variable corresponde a la dirección del *dataset* que se va a utilizar para ejecutar alguno de los algoritmos. Este parámetro también es obligatorio.
- **Carga:** Es el valor de la carga para el algoritmo PPPL. Este parámetro no es obligatorio. Si no se ingresa un valor, se asigna el valor 0 para la carga.

Para mayor detalle respecto al funcionamiento de la librería, se aconseja revisar el enlace del repositorio en Github.

4.2. Definición Campos Resultados

Los campos que se utilizaron para guardar los resultados obtenidos de la ejecución de los diferentes algoritmos bajo los diferentes *datasets* son los siguientes:

- **Tiempo de Ejecución:** Este valor corresponde al tiempo cronometrado de la ejecución del algoritmo bajo un *dataset*. Este valor está medido en segundos y considera desde la carga del *dataset* hasta antes de la escritura en disco de los resultados obtenidos.
- **Número de Clausuras Calculadas:** Este campo corresponde a la cantidad de veces que el algoritmo calculó la clausura para una combinación de atributos determinada. En el caso de los algoritmos paralelos, se considera como la suma total de las clausuras calculadas en todos los procesos.
- **Memoria Utilizada:** Corresponde a la cantidad de memoria utilizada por el algoritmo durante su ejecución. En los algoritmos paralelos, corresponde a suma de la memoria utilizada por todos los algoritmos.

4.3. Resultados Obtenidos PPPL

A continuación, se presentan las tablas y gráficos de los resultados obtenidos de la ejecución del algoritmo PPPL bajo diferentes cargas en los *datasets* seleccionados que fueron nombrados anteriormente. Se destaca con azul aquel resultado que obtuvo el mejor desempeño por cada *dataset*.

Tabla 6: Tiempos de ejecución algoritmo PPPL a diferentes cargas.

Dataset	5 [s]	8 [s]	12 [s]	15 [s]
Adult	3.2522	3.4667	11.6124	50.9355
Car	184.7123	343.3422	361.2950	359.1791
Credit	76.6892	174.4477	1408.9617	3913.6733
Flight	2453.8300	2496.8921	2548.8044	2451.5614
Horse	2629.4565	1117.2725	1432.9785	3814.5247
Mushroom	337.5264	103.4225	81.6539	277.4066
Servo	105.3634	105.4451	104.6222	105.9188

Tabla 7: Número de clausuras calculadas algoritmo PPPL a diferentes cargas.

Dataset	5	8	12	15
Adult	16944	17664	26624	104076
Car	176	448	448	448
Credit	37667	38387	58041	119745
Flight	2696578	2696578	2696578	2696578
Horse	3300060	3300364	3306864	3337268
Mushroom	1904302	1904606	1913502	2001722
Servo	208204	208204	208204	208204

Tabla 8: Memoria utilizada algoritmo PPPL a diferentes cargas.

Dataset	5 [Mb]	8 [Mb]	12 [Mb]	15 [Mb]
Adult	90.5156	93.1445	97.4296	127.6992
Car	1455.8789	1457.9765	1457.8906	1458.1523
Credit	362.2109	364.0039	366.3125	374.9296
Flight	2774.8554	2775.7929	2776.0898	2776.0117
Horse	1769.5703	1031.3515	869.2265	871.9257
Mushroom	793.8320	694.4101	700.2773	748.8945
Servo	269.0976	268.2109	269.0312	268.8437

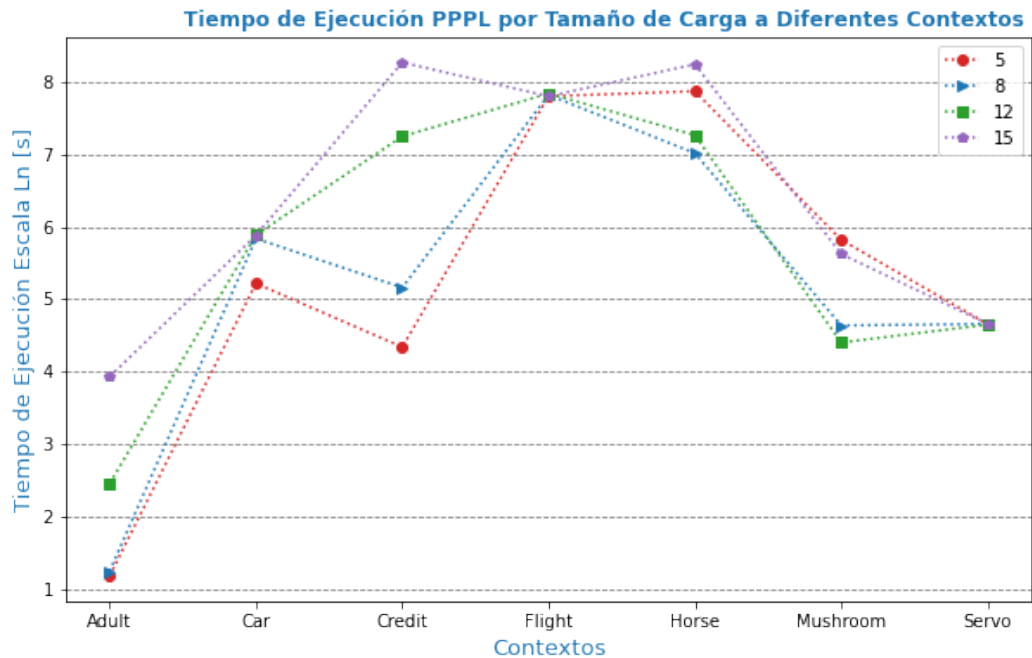


Figura 6: Tiempo de ejecución en PPPL por Tamaño de Carga en Diferentes Contextos

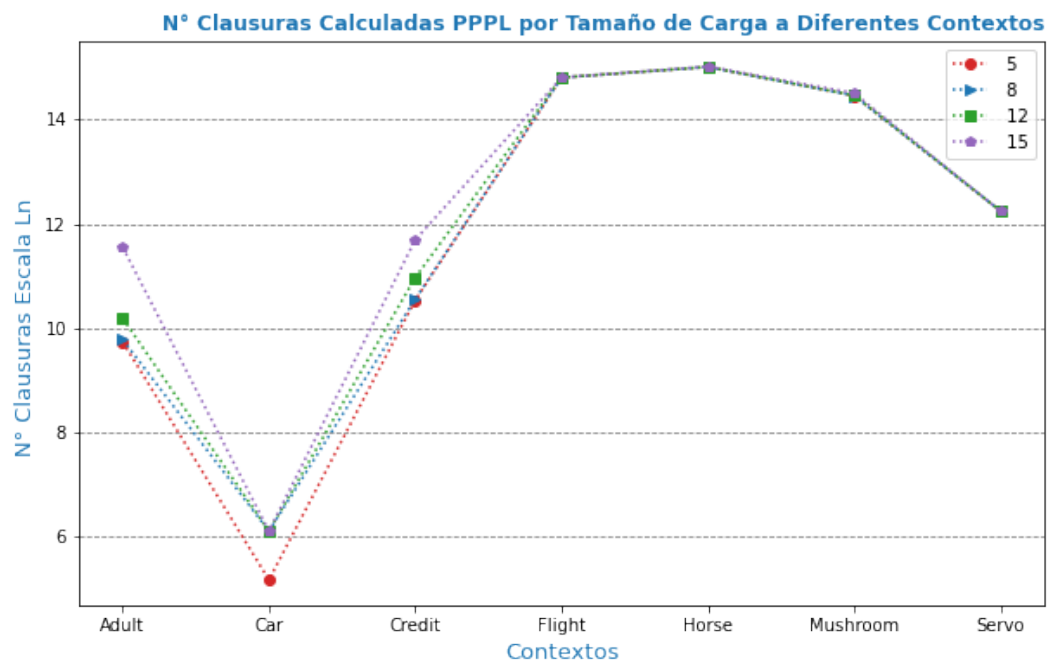


Figura 7: Clausuras calculadas en PPPL por tamaño de carga en diferentes contextos

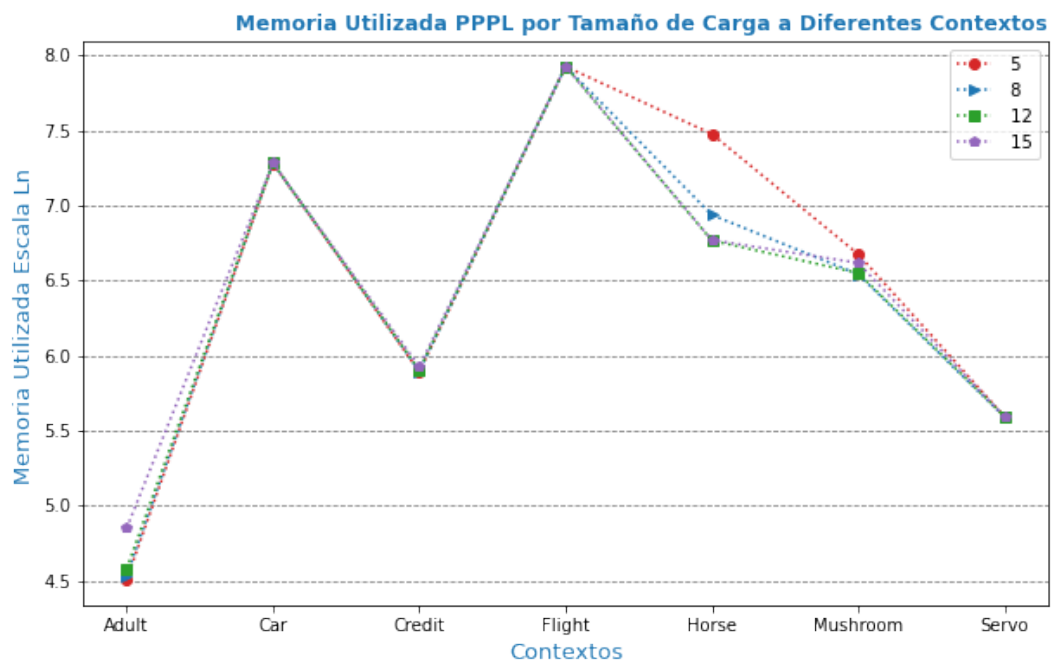


Figura 8: Memoria utilizada en PPPL por tamaño de carga en diferentes contextos

A partir de estos resultados, se decide seleccionar para la comparación general de los algoritmos, los resultados de la ejecución de PPPL bajo la carga que obtuvo mejores resultados para cada uno de los *dataset*. Es por esto que se decidió considerar para Adult, Car y Credit los resultados de PPPL con carga igual a 5; para Flight, Mushroom y Servo con carga igual a 12, y, para Horse PPPL con carga igual a 8.

4.4. Resultados Generales

En esta sección se presentan los resultados obtenidos de la ejecución todos los algoritmos, incluyendo los resultados del algoritmo PPPL bajo la carga que obtuvo el mejor desempeño determinada en la sección anterior. Las tablas y gráficos en detalle por *dataset* se encuentran en la sección de anexos de este documento.

A continuación, se presentan las tablas y gráficos que resumen los resultados obtenidos por los algoritmos para cada *dataset* en cada uno de los campos. En cada fila de las diferentes tablas, se distingue con rojo aquel resultado que fue el peor desempeño para dicho *dataset* y con azul el que obtuvo el mejor. Cabe destacar, que, en tiempo de ejecución, número de clausuras calculadas y memoria utilizada, un mayor número se traduce como un peor desempeño en cada categoría.

Tabla 9: Resultados tiempos de ejecución por algoritmos.

Dataset	Apriori [s]	Apriori Paralelo [s]	PPPL [s]	SPPL [s]	NextClosure [s]
Adult	8.0558	7.7237	3.2522	1.8284	0.1842
Car	113.4205	102.7258	184.7123	16.9712	7.5078
Credit	5231.9614[s]	649.3346	76.6892	7.4807	5.3883
Flight	3559.2805	912.6865	2451.5614	14.4910	27.5380
Horse	34320.0000	14571.9325	1117.2725	204.5191	164.3000
Mushroom	3559.2805	912.6865	81.6539	14.4910	27.5380
Servo	42.3018	37.0404	104.6222	108.3769	172.4305

Tabla 10: Resultados número de clausuras calculadas.

Dataset	Apriori	Apriori Paralelo	PPPL	SPPL	NextClosure
Adult	200340	200340	16944	16890	16896
Car	889	889	176	124	128
Credit	232944	232944	37667	37608	37619
Flight	21409343	21409343	2696578	1904282	1904286
Horse	27246689	27246689	3300364	3300006	3300044
Mushroom	21409343	21409343	1913502	1904282	1904286
Servo	231680	231680	208204	293760	208205

Tabla 11: Resultados memoria utilizada.

Dataset	Apriori [Mb]	Apriori Paralelo [Mb]	PPPL [Mb]	SPPL [Mb]	NextClosure [Mb]
Adult	21.7265	124.6367	90.5156	84.1992	15.3125
Car	527.9687	578.3320	1455.8789	610.1835	632.8828
Credit	182.7656	491.8515	362.2109	264.8046	205.9335
Flight	788.5039	2391.7890	2776.0117	237.4609	202.875
Horse	-	3644.5664	1031.3515	332.4453	312.0234
Mushroom	788.5039	2391.7890	700.2773	237.4609	202.875
Servo	18.9648	68.9179	269.03125	87.1835	17.9140

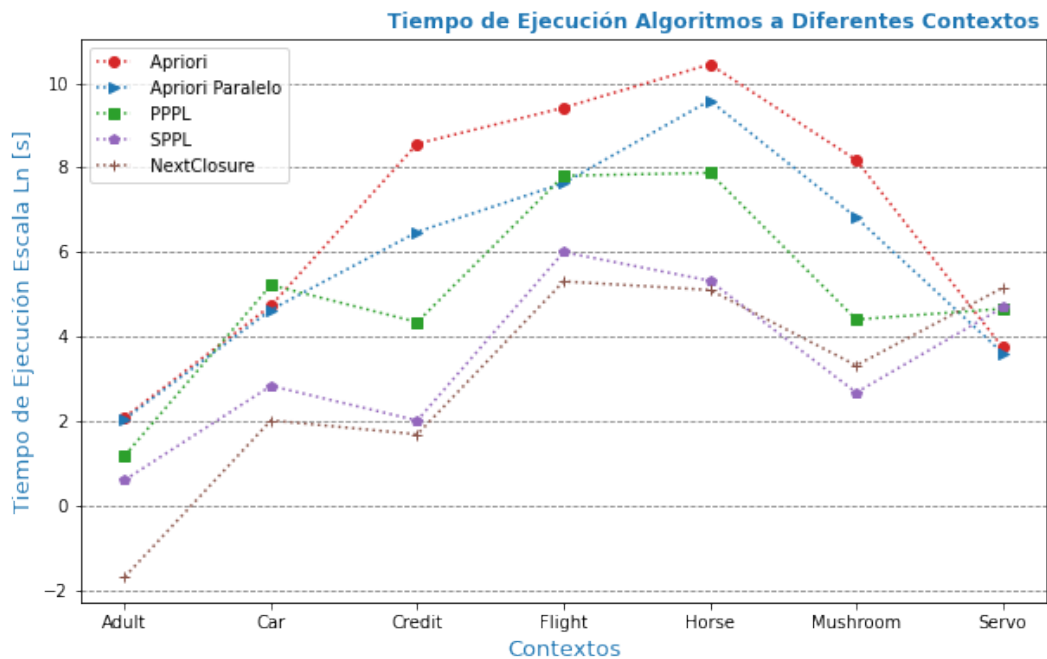


Figura 9: Resultados tiempos de ejecución algoritmos a diferentes contextos

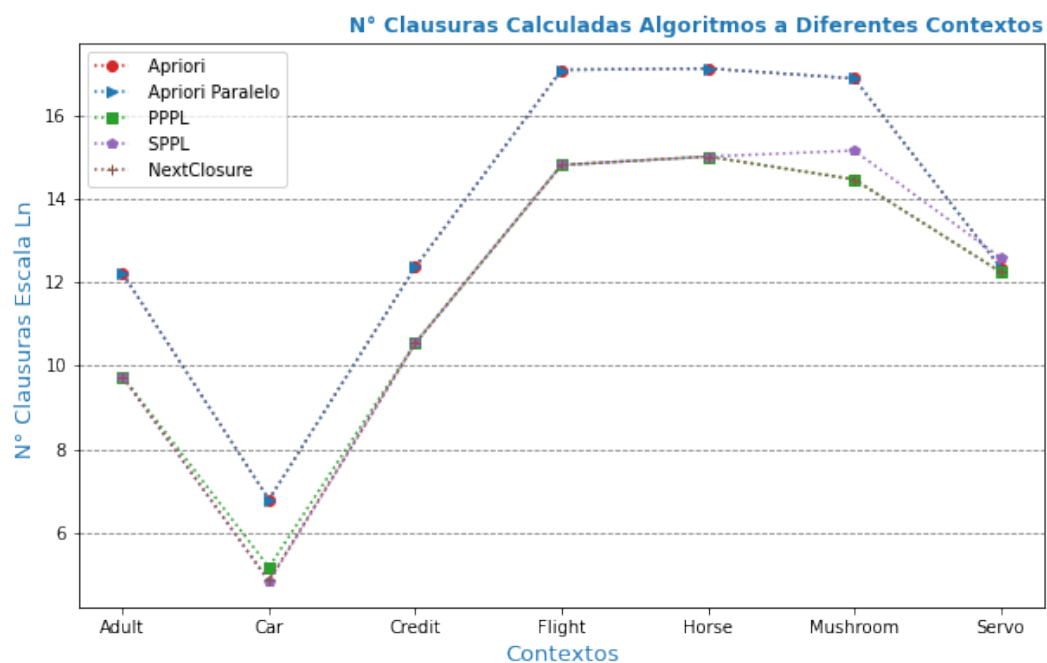


Figura 10: Resultados clausuras calculadas algoritmos a diferentes contextos

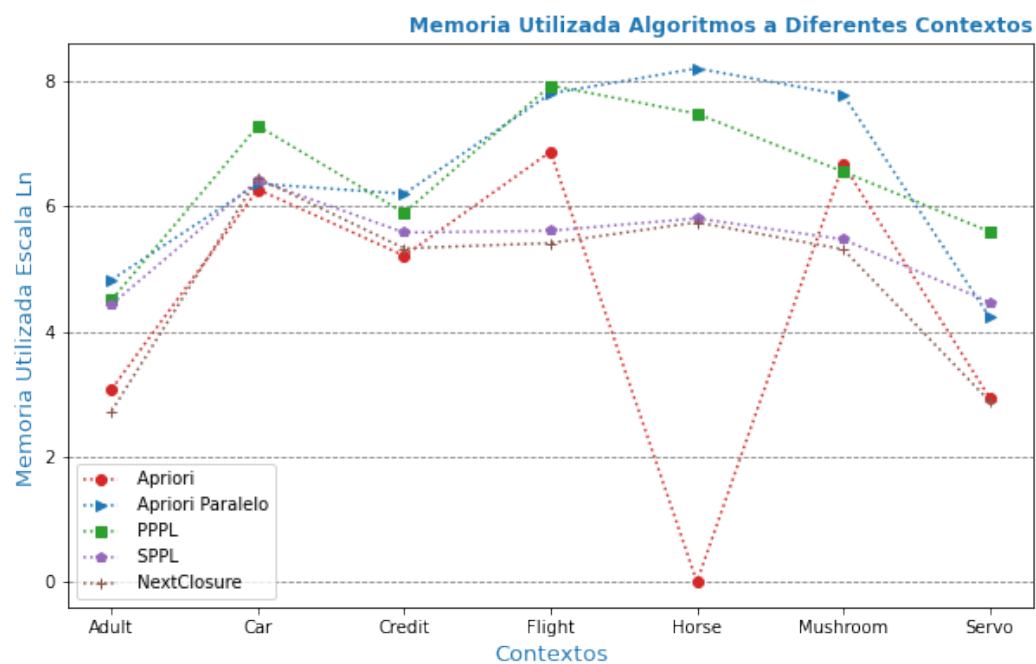


Figura 11: Resultados memoria utilizada algoritmos a diferentes contextos

4.5. Análisis resultados

Como se mencionó anteriormente, la complejidad de la tarea de encontrar los conceptos formales radica en la necesidad de calcular las clausuras de las diferentes combinaciones de atributos o ejemplos. Un algoritmo que calcule más clausuras innecesarias tendrá, lógicamente, un tiempo de ejecución mayor que uno que calcule menos. Esta premisa se propone a partir de los resultados en las tablas 9 y 10, donde, en la mayoría de los *datasets*, los algoritmos que más clausuras calcularon fueron también aquellos que más tiempo demoraron en su ejecución. Cabe destacar que en Apriori y Apriori paralelo, se calcularon la misma cantidad de clausuras debido a que la diferencia existente entre ambos algoritmos es que uno ejecuta el proceso de cálculo de clausuras en paralelo, mientras que el otro ejecuta el proceso de forma secuencial. Sin embargo, este último tuvo mejores resultados en tiempo, lo que debe al beneficio del paralelismo en el cálculo de clausuras. Gracias a este beneficio, la hipótesis de que un algoritmo calcule mayor cantidad de clausuras innecesarias ocupará más tiempo que otro que no lo haga, no se cumplirá necesariamente en todos los casos si se trata de algoritmos paralelos y secuenciales al mismo tiempo. Esto se debe a que un algoritmo que calcule más clausuras innecesarias, pero de forma paralela, podría tener mejores resultados que uno que calcule menos, pero de manera secuencial. Lo expuesto, se observa en la tabla 29 para el *dataset* Servo, donde los resultados exponen que el algoritmo Apriori Paralelo obtuvo un mayor número de clausuras calculadas que NextClosure, pero Apriori Paralelo logra un mejor resultado en tiempo de ejecución.

Respecto al uso de memoria por parte de los algoritmos, se observa que aquellos algoritmos que están paralelizados, hacen mayor uso de memoria independiente del enfoque que utilizan (*depth-first* o *breadth-first*). Este aumento de uso de memoria de los algoritmos se puede atribuir al *Scheduling* de dichos procesos. Sin embargo, se destaca que a partir de los resultados de la tabla 11, la paralelización de los algoritmos Apriori y PPPL, que son algoritmos que emplean el patrón Maestro/Esclavo, utilizan más memoria que SPPL, cuyo patrón se basa en Dividir y Conquistar.

Al analizar los resultados del algoritmo NextClosure, se observa el impacto del test canónico, ya que evita el cálculo de una gran cantidad de clausuras innecesarias. Esta cualidad se evidencia al comparar la cantidad de cálculos de clausuras en la tabla 10, con respecto a los resultados de Apriori y Apriori Paralelo, algoritmos que no poseen el test canónico, donde la diferencia es cercana a un orden de magnitud.

Junto con lo anterior, en los resultados de la tabla 10 respecto al número de clausuras calculadas en los diferentes contextos, PPPL obtuvo resultados muy similares a los algoritmos NextClosure y SPPL, siendo este último quien obtuvo la menor cantidad de clausuras calculadas en la mayoría de los *datasets*. Sin embargo, se advierte que no ocurre lo mismo respecto al tiempo de ejecución. Aunque los valores respecto al tiempo de ejecución para esta propuesta son más cercanos a NextClosure y SPPL que a las dos versiones del algoritmo Apriori, la diferencia de los resultados de PPPL y los dos primeros algoritmos es de prácticamente

un orden de magnitud. Esta diferencia es algo muy llamativo debido a la similitud en la naturaleza entre estos algoritmos, por lo que, estudiando y comparando ambos algoritmos, se barajan las siguientes posibles razones de esta diferencia en los tiempos de ejecución:

- **Carga:** Un problema importante es el cómo determinar cuál es la carga óptima para cada *dataset*. Como se observa en los resultados de PPPL, por ejemplo en la tabla 21 y la tabla 22, los tiempos de ejecución pueden variar en un orden de magnitud entre un par de valores determinados para la carga. Esto último, demuestra la necesidad de generar un estudio respecto a la existencia de una relación entre el número de atributos del *dataset* y la carga óptima para la ejecución de esta propuesta PPPL.

Observando los resultados de la tabla 22, se puede notar que no hay una relación trivial entre el tamaño de la carga y el tiempo de ejecución, por lo que, resolver el problema de determinar la carga óptima para un contexto formal determinado, podría ser tan complejo como ejecutar ese *dataset* con diferentes valores predeterminados de carga.

Como último punto, la carga es un valor constante durante toda la ejecución de PPPL siguiendo un enfoque de balanceo de carga estático. Por este motivo, es una alternativa de estudio replantear el rol de la carga a un enfoque de balanceo de carga dinámico cambiando el valor de la carga en cada nivel.

- **Bloques desbalanceados:** Debido a la característica *breadth-first* de PPPL, al paralelizar los bloques de cada nivel, se obtienen casos donde diferentes bloques tienen una cantidad de clausuras que calcular bastante desiguales.

Para un mismo nivel, un bloque generado por un candidato, ejecutará la búsqueda de conceptos formales guiado por el orden de la enumeración léctica. Si este bloque, aprovechando la propiedad del test canónico, puede evitar hacer el cálculo de todas las clausuras de su bloque, este proceso terminará rápidamente, mientras que otros procesos que están ejecutando otros bloques siguen trabajando.

Comparando el tiempo de ejecución y el número de clausuras calculadas por PPPL y NextClosure en las tablas 27 y 28, el número de clausuras calculadas por ambos son muy cercanos, pero los tiempos de ejecución no. En este caso, se podría sospechar que, dado estos *dataset* y la naturaleza del algoritmo, se estaría generando en algunos niveles el problema de desbalance entre los bloques. De esta manera, mientras el algoritmo NextClosure avanza de manera secuencial aprovechando la particularidad del test canónico, PPPL debe sumar en cada nivel de ejecución el tiempo que tarda el bloque que debe realizar el mayor número de clausuras y además el *Overhead* relacionado al *Scheduling* de los bloques del nivel. De esta manera, este desbalance podría ser uno de los motivos por los que el algoritmo tarda más que NextClosure.

- **Secuencialidad por niveles:** Debido a que la importancia de los niveles radica en que en estos se produce el cálculo de las clausuras de una combinación compuesta por i atributos más la carga, con i igual al número del nivel, la secuencialidad de los niveles no es un problema en propiamente tal, sino que este radica en que no se puede ejecutar ningún cálculo de algún bloque del siguiente nivel sin que antes termine la ejecución de todos los procesos que están calculando los bloques del nivel anterior.

Es por esto, que replantear la paralelización de PPPL de una paralelización en cada nivel a dividir el trabajo del cálculo de clausuras en diferentes procesos que vayan trabajando en diferentes regiones de la enumeración léctica podría ser un enfoque interesante a estudiar. Al dividir el trabajo por zonas del árbol léctico, como lo hace SPPL, permitiría paralelizar en diferentes cantidades de procesos las diferentes regiones del árbol. Sumado a esto, este proceso podría realizarse de manera dinámica, vale decir, si hay una zona de la enumeración léctica que termina antes su ejecución de otra, podría asignársele parte del trabajo del cálculo de clausuras de un proceso que aún no termina, paralelizando el trabajo de manera dinámica y buscando resolver el problema del desbalanceo asignándole trabajo a los procesos que ya terminaron.

Continuando con el análisis de los resultados obtenidos, se observa que los algoritmos NextClosure y SPPL los resultados son bastante variados. La capacidad de paralelizar el enfoque *depth-first* y las ventajas del test canónico en SPPL logran ser una buena vía para competir con el desempeño de NextClosure. Además de esto, se destaca que en los *datasets* Flight y Mushroom SPPL supera en desempeño a NextClosure. Junto con esto, en los resultados en la tabla 9, la diferencia entre ambos algoritmos es pequeña en términos cuantitativos considerando las magnitudes de las diferencias que existen entre NextClosure con el resto de los algoritmos implementados, y sumado al tiempo de *Overhead* que se le suma a SPPL al ser un algoritmo paralelo.

Comparando los resultados obtenidos por las propuestas PPPL y SPPL en la tabla 9, la segunda propuesta obtuvo mejores resultados. Esta gran diferencia entre ambos algoritmos se asemeja mucho a las problemáticas que se detallan entre PPPL y NextClosure. Se puede notar que la SPPL no sufre de la secuencialidad por niveles debido a su capacidad de avanzar de manera asíncrona por diferentes ramas de trabajo del contexto formal. Debido a esto, no sufre los efectos de los bloques desbalanceados, ya que no se necesita que terminen todos los bloques de un nivel para ejecutar los siguientes, sino que se duplica los procesos trabajando en un subtarea si es que esta demora mucho en su ejecución. Cabe destacar que, dado que la carga es un concepto solamente de PPPL, SPPL no sufre respecto a la determinación del valor de la carga ya que no usa este parámetro en para su ejecución.

Por último, se destacan los resultados obtenidos en bajo el *dataset* Servo Invertido que se observan en la tabla 29. Dicho *dataset* que posee solamente cinco objetos pero, contiene 7240 atributos. Este *dataset* fue el único escenario donde los algoritmos con un enfoque

breadth-first obtuvieron mejores resultados que el resto. Una hipótesis válida para explicar estos resultados podría ser la existencia de una relación entre el desempeño de un algoritmo con enfoque *breadth-first* bajo contextos en el que hay una poca cantidad de objetos y una gran cantidad de de atributos. Debido a lo llamativo de estos resultados, se extendió el estudio para analizar este escenario.

4.6. Estudio Caso Dataset Servo

En esta sección se realiza un análisis entre el algoritmo Apriori y el algoritmo NextClosure, algoritmos de enfoques de búsqueda diferentes, para buscar explicar los resultados obtenidos en el *dataset* Servo. Para este análisis, se modificó el código de ambos algoritmos, permitiendo registrar el tiempo utilizado por los algoritmos a lo largo de las diferentes secciones de estos. Además, se registra el tiempo de ejecución por nivel en el algoritmo Apriori y el tiempo que se demora el algoritmo NextClosure en encontrar cada concepto formal. Junto con esto, en pro de sumar información a las características de cada algoritmo, se registran la cantidad de candidatos que revisa Apriori en cada nivel y la cantidad de clausuras que calcula NextClosure en encontrara cada concepto formal.

El entorno de ejecución para este análisis fue en un computador Intel I7-4470 CPU@3.24GHz, con 8 núcleos, con 20 GB RAM, con sistema operativo Ubuntu 20.04.5 LTS. Para este análisis se eligió el *Servo* para estudiar el comportamiento de los algoritmos bajo este contexto. Junto con esto, para contrastar algunas posibles hipótesis se eligió invertir el contexto *Servo* generando un *dataset* que posee una gran cantidad de objetos pero una pequeña cantidad de atributos. Junto con estos dos, se eligió el *dataset* Adult y su versión invertida que permite tener un desbalance entre objetos y atributos pero no tan desproporcionado como los dos primeros *datasets*. Las características de los contextos elegidos se presentan a continuación:

Tabla 12: Datasets seleccionados

Nombre	Número de objetos	Número de atributos	Número de Conceptos Formales
Adult	388	14	13357
Adult Invertido	14	388	13357
Servo	5	7240	32
Servo Invertido	7240	5	32

A continuación, se presentan los resultados medidos para Apriori y NextClosure. Se destaca con azul aquel resultado que fue el mejor en cada categoría.

Tabla 13: Resultados tiempos de ejecución Apriori y NextClosure

Nombre	Apriori [s]	NextClosure [s]
Adult	5.5326	0.1549
Adult Invertido	131.4686	145.9907
Servo	33.6336	129.6602
Servo Invertido	0.0784	0.03354

Tabla 14: Resultados clausuras calculadas Apriori y NextClosure

Nombre	Apriori	NextClosure
Adult	200340	16896
Adult Invertido	5182128	4048075
Servo	231680	208205
Servo Invertido	155	32

Tabla 15: Resultados memoria utilizada Apriori y NextClosure

Nombre	Apriori [Mb]	NextClosure [Mb]
Adult	22.3007	18.8046
Adult Invertido	40.8359	180.4531
Servo	19.375	25.9570
Servo Invertido	15.13671	16.4765

A continuación, se presentan tres gráficos por *dataset*. De estos gráficos, el primero busca mostrar el comportamiento de ambos algoritmos a lo largo de su ejecución respecto a cuánto tiempo tardan en las diferentes secciones del algoritmo. Para el caso de Apriori, estas secciones es el tiempo ocupado en buscar nuevos conceptos formales a partir de los diferentes candidatos, mientras que para NextClosure es el tiempo que tarde en encontrar cada concepto formal. Cabe destacar, que, para esta primera gráfica se presenta una parte de los tiempos más altos registrados en pro de poder mostrar el comportamiento de ambos algoritmos a lo largo de su ejecución.

El segundo gráfico por cada *Dataset* es la cantidad de candidatos que revisa el algoritmo Apriori por nivel. En esta gráfica, también se presenta una porción de los valores registrados, mostrando siempre los valores más altos. Por último, el tercer gráfico muestra la cantidad de clausuras calculadas por concepto formal a lo largo de la ejecución del algoritmo Next-Closure, también considerando los valores más altos para permitir un entendimiento de la imagen.

4.6.1. Dataset Servo

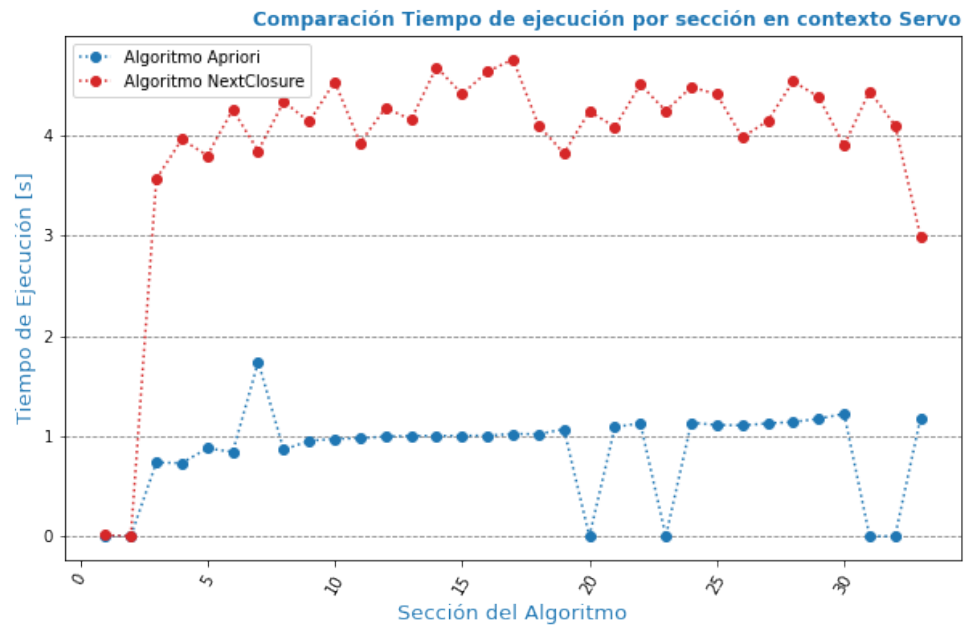


Figura 12: Tiempo de ejecución por sección de los algoritmos en contexto Servo.

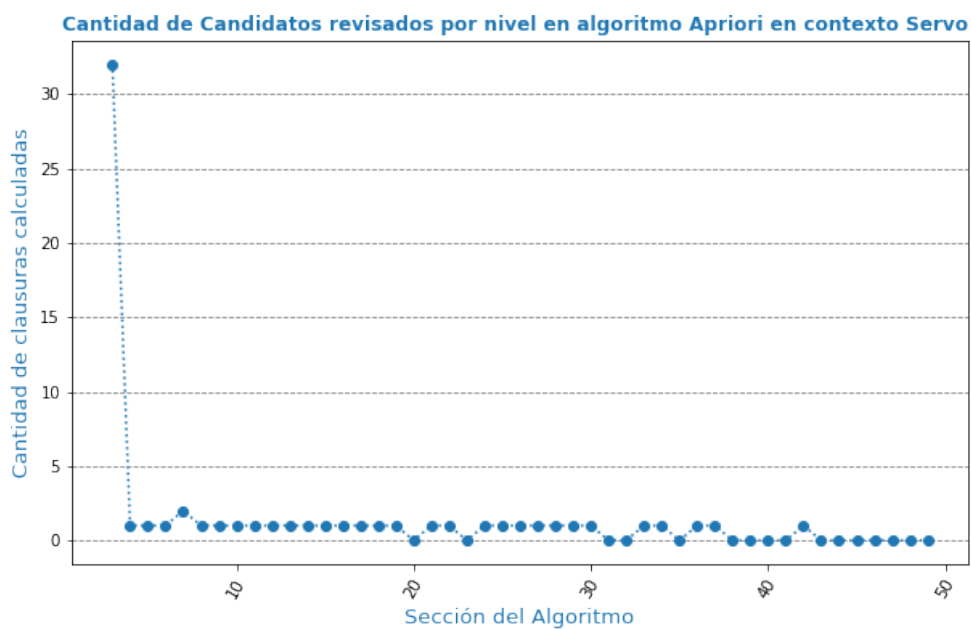


Figura 13: Candidatos revisados por nivel en Apriori en contexto Servo.

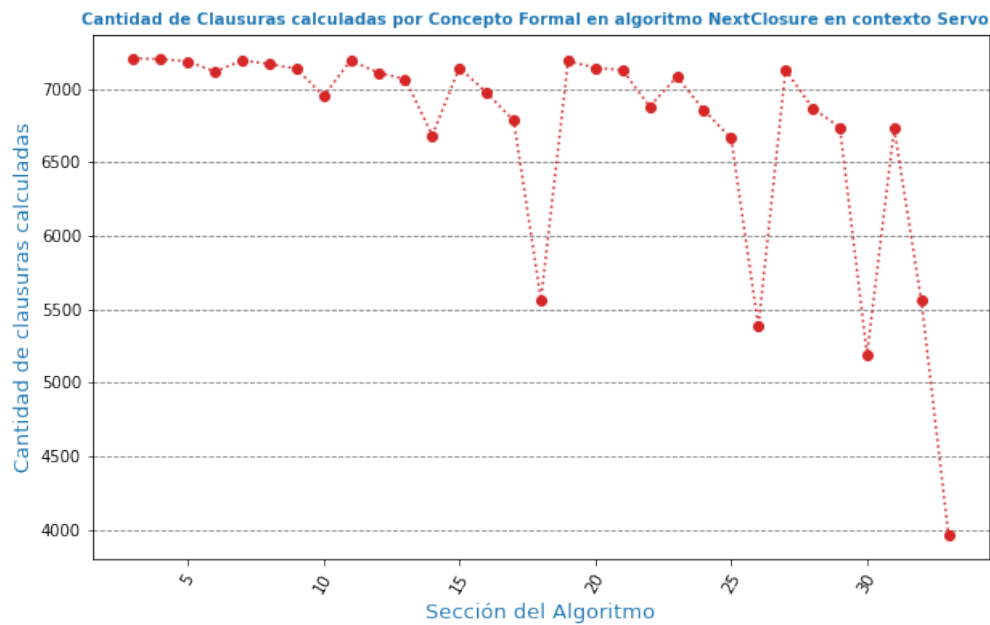


Figura 14: Clausuras calculadas por concepto formal en NextClosure en contexto Servo.

4.6.2. Dataset Servo Invertido

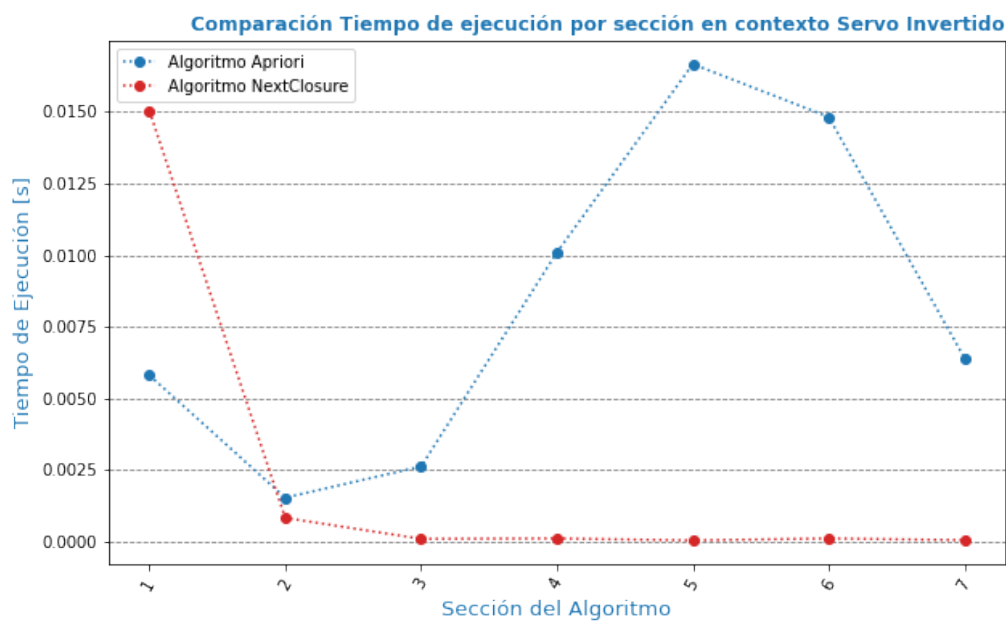


Figura 15: Tiempo de ejecución de los algoritmos en contexto Servo Invertido.

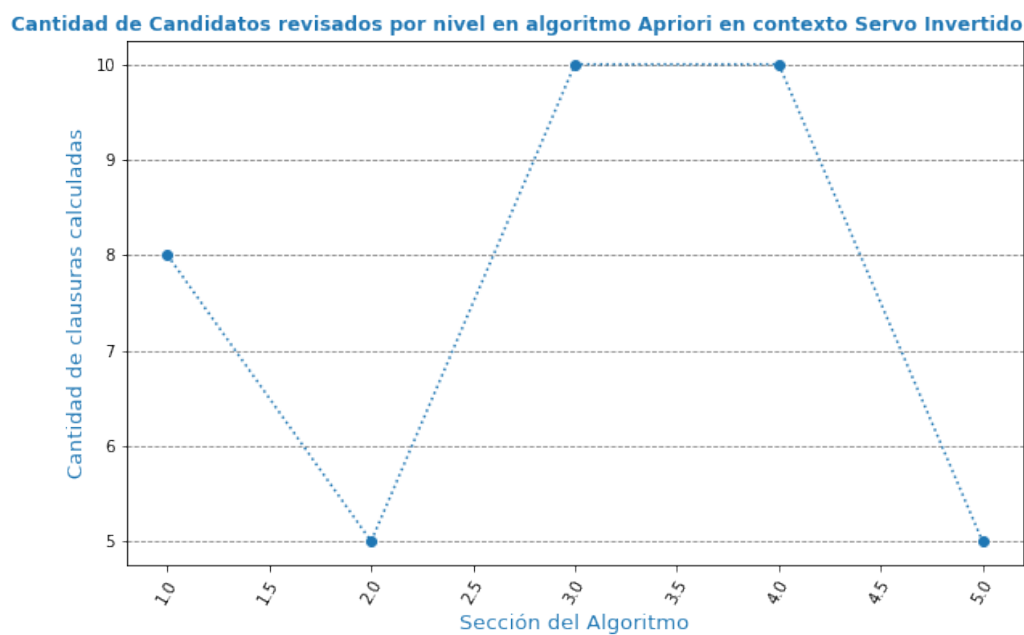


Figura 16: Candidatos revisados por nivel en Apriori en contexto Servo Invertido.

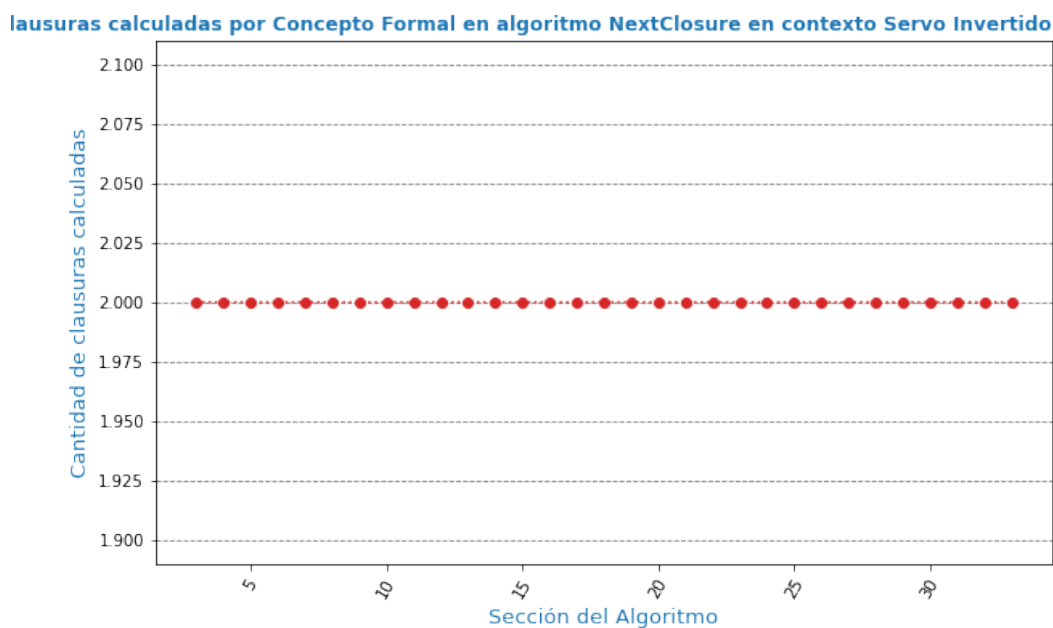


Figura 17: Clausuras calculadas por concepto formal en NextClosure en contexto Servo Invertido.

4.6.3. Dataset Adult

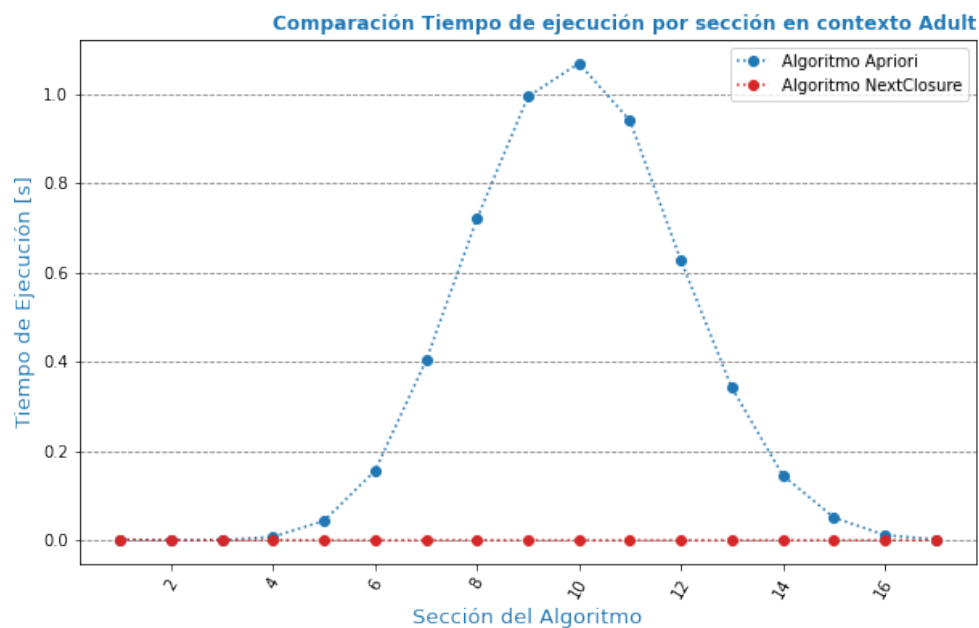


Figura 18: Tiempo de ejecución de los algoritmos en contexto Adult.

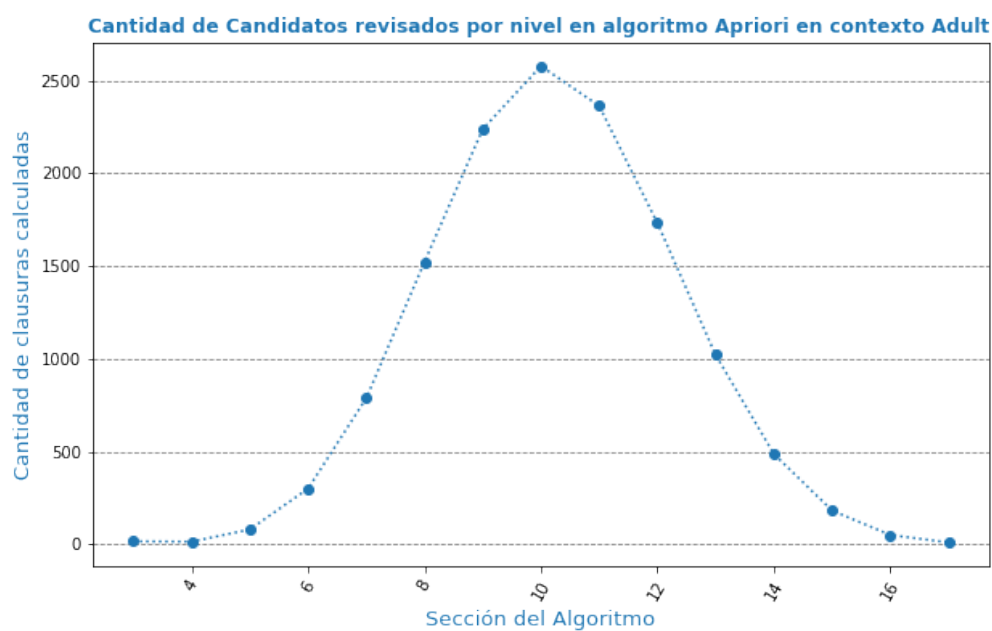


Figura 19: Candidatos revisados por nivel en Apriori en contexto Adult.

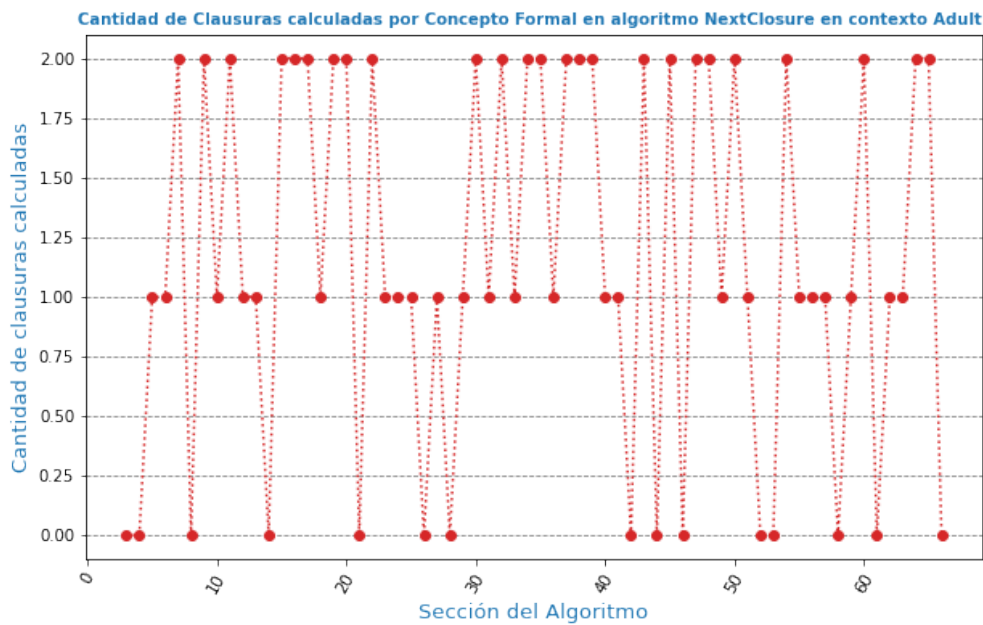


Figura 20: Clausuras calculadas por concepto formal en NextClosure en contexto Adult.

4.6.4. Dataset Adult Invertido

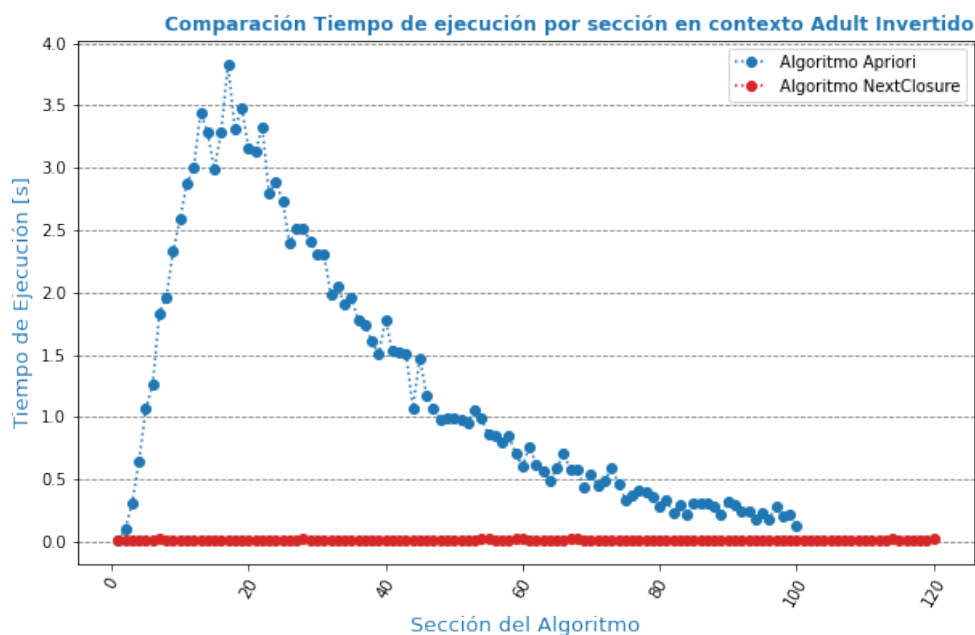


Figura 21: Tiempo de ejecución de los algoritmos en contexto Adult Invertido.

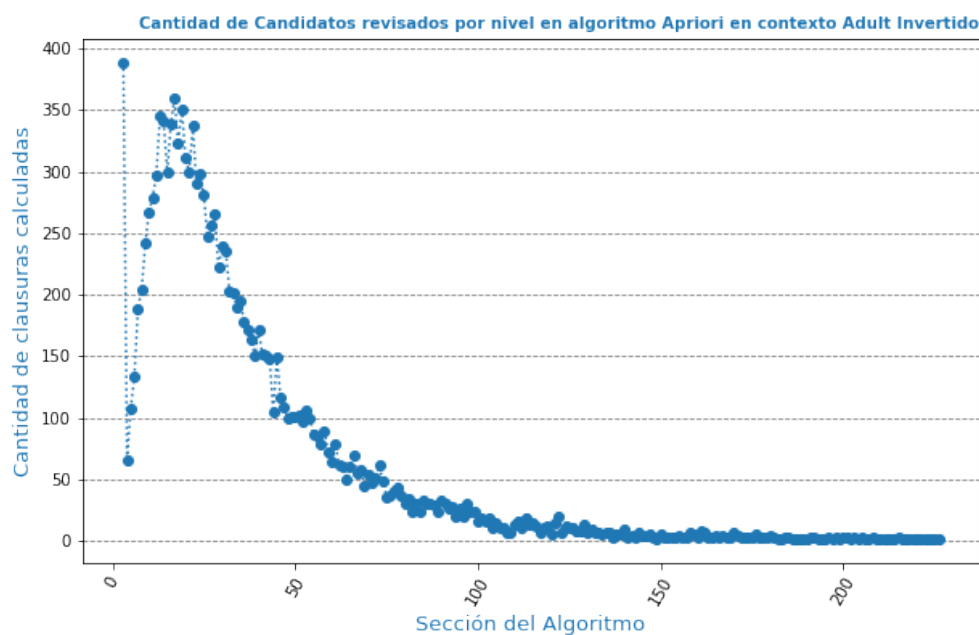


Figura 22: Candidatos Revisados por nivel en Apriori en contexto Adult Invertido.

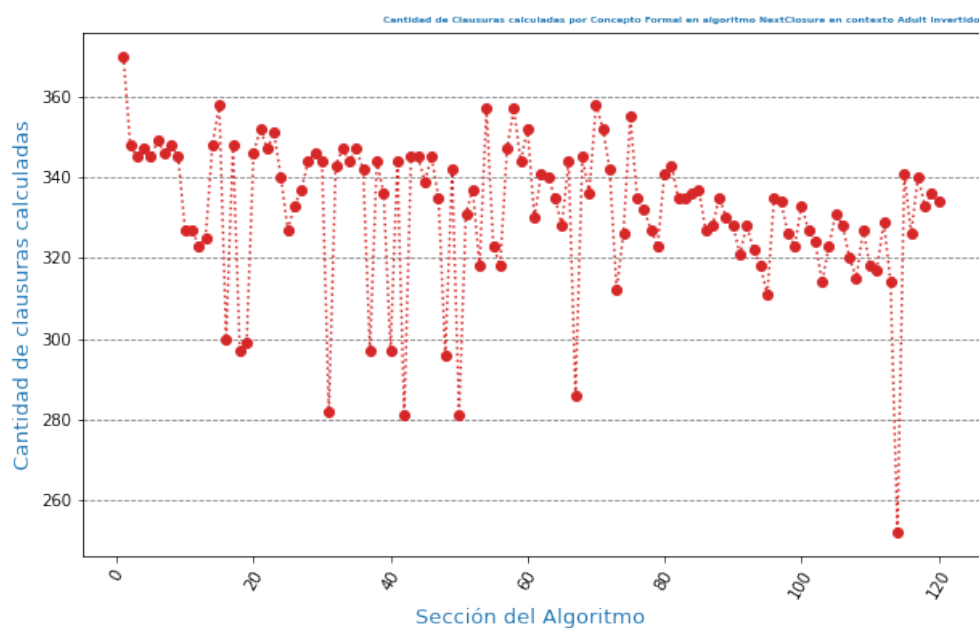


Figura 23: Clausuras calculadas por concepto formal en NextClosure en Adult Invertido.

4.7. Análisis Resultados Caso Dataset Servo

Ante los resultados encontrados en la tabla 9 sobre el comportamiento de los algoritmos en el *dataset* Servo, se puede percatar que es el único contexto bajo el cual el algoritmo Apriori y su versión paralela tiene un mejor desempeño que Nextclosure y SPPL, siendo estos dos últimos aquellos con mejores resultados en el resto de los *dataset*. Una característica diferenciadora de Servo y los otros contextos, es que en este hay una gran cantidad de atributos y una ínfima cantidad de objetos. Es por esto que, ante la sorpresa de estos resultados, se ejecuta Apriori y NextClosure bajo los *dataset* explicados en la sección anterior. Estos resultados que se encuentran en la tabla 13 nos muestra que al invertir Servo, es decir, queda con muchos más objetos que atributos, el desempeño de los algoritmos se invierte y NextClosure vuelve a tener un mejor resultado. Sin embargo, gracias a que se invirtió el *dataset* Adult, de manera que también se tiene un contexto que tiene una mayor cantidad de atributos que de objetos, Apriori nuevamente obtiene un mejor resultado, no tan holgado como en el caso de Servo, pero sí un mejor desempeño. Cabe destacar que Adult invertido no tiene una proporción entre atributos y objetos tan desigual como la tiene el *dataset*.

Observando el gráfico 12 se puede observar que el algoritmo NextClosure demora mucho más tiempo en encontrar cada concepto formal de lo que Apriori tarda en revisar los candidatos de cada nivel. Viendo también el gráfico 13, se puede notar que el algoritmo Apriori logra encontrar los 32 conceptos formales en la primera sección del código, que trata sobre calcular la clausura de cada atributo. De esta manera, a lo largo de la ejecución del algoritmo, los únicos candidatos que revisa son los 32 conceptos formales que ya se calcularon. Esta propiedad permite evitar revisar candidatos de algunos niveles debido a que ya tiene los conceptos formales. Sin embargo, esta característica no evita que vuelva a calcular clausuras a partir de los candidatos en búsqueda de posibles nuevos conceptos formales. Por otro lado, del gráfico 14 se puede observar que NextClosure debe calcular muchas clausuras para encontrar los primeros conceptos formales, de hecho, debe llegar a una calcular más de 7000 clausuras siendo que este contexto posee 7240 atributos. Esta característica podría responder al comportamiento que tiene NextClosure que se ve en el gráfico 12 donde claramente se evidencia que tarda bastante más que Apriori.

Este comportamiento encontrado de Apriori en el escenario de Servo se repite en parte en el gráfico 22, donde se puede ver que Apriori identifica casi 400 conceptos formales en la derivación de cada atributo y el resto los logra identificar en los niveles más próximos, haciendo que en los niveles más altos deba revisar una menor cantidad de candidatos. Ya que si se tiene X el conjunto de atributos de la combinación a revisar, G el conjunto de objetos del contexto formal y $\varphi(g)$ los atributos de un objeto g , el operador de clausura debe revisar por cada $g \in G$ si $X \subseteq \varphi(g)$. De esta manera, a un X de mayor tamaño, la verificación de la relación de subconjunto tardará más. Es por esto, considerando los resultados en 13, uno de los motivos que provocan que Apriori tarde menos que NextClosure en Adult Invertido, puede ser que Apriori debe revisar una menor cantidad de candidatos

de mayor cantidad de atributos en los niveles más altos. Mientras tanto, en Nextclosure, viendo el gráfico 23, la cantidad de clausuras que calcula por cada concepto formales tiende a disminuir levemente. Cabe destacar que, aunque en el gráfico 21 Apriori tarda más en las primeras secciones o primeros niveles durante su ejecución, NextClosure tiene un tiempo menor y constante entre cada concepto formal pero tarda más en terminar de encontrar todos los conceptos formales.

Desde otro punto, observando lo que ocurre cuando se trabaja bajo Servo Invertido en el gráfico 16, Apriori obtiene una pequeña cantidad de conceptos formales al calcular las clausuras de cada atributo, y concentra su ejecución en los niveles intermedios donde trata con una mayor cantidad de candidatos. Mientras tanto, se puede observar en la figura 17, que NextClosure calcula una muy baja cantidad de clausuras por cada concepto formal. Sumado a esto, en el gráfico 15 y los resultados en 13, se puede evidenciar que NextClosure tarda una menor cantidad de tiempo en encontrar todos los conceptos formales debido a lo mencionado anteriormente.

Esto también sucede observando el comportamiento de ambos algoritmos en el *dataset* Adult, donde en el gráfico 19 se puede observar que Apriori tiene una mayor concentración de candidatos a revisar en los niveles intermedios de la ejecución y obteniendo pocos conceptos formales al inicio de su ejecución, en el cálculo de la clausura de los atributos. Por otro lado, en la figura 20, se puede observar que NextClosure calcula una cantidad baja y sostenida de clausuras por cada concepto formal. Las consecuencias de esto se pueden notar en el gráfico 18, donde se evidencia que la concentración de candidatos de Apriori genera una mayor demora en los niveles intermedios de su ejecución mientras NextClosure mantiene un tiempo bajo y constante durante su ejecución, logrando un mejor desempeño.

Por último, cabe mencionar que se revisó que ambos algoritmos ocuparan la misma implementación (o equivalente algorítmico) respecto del operador de clausura. De todo lo anterior, se puede desprender que la presencia de una menor cantidad de candidatos en los niveles intermedios y altos durante la ejecución logra mejores resultados, y este escenario se da en los *datasets* que tienen una mayor cantidad de atributos que de objetos, por lo que podría haber una relación existente entre ambas condiciones. Sin embargo, aunque Apriori tuvo un mejor desempeño que NextClosure en Servo y Adult Invertido, los tiempos de ejecución fueron mucho mejores en NextClosure con Servo Invertido y Adult, donde la diferencia de los tiempos de ejecución para los mismos *datasets* contra su versión invertida es de más de un orden de magnitud. Es por esto que el desempeño que logró Apriori en escenarios donde existe una mayor cantidad de atributos, no logra superar los resultados de NextClosure en contextos con una mayor cantidad de objetos.

CAPÍTULO 5

CONCLUSIONES

En este capítulo, se presentan las conclusiones desde diversos tópicos, abarcando desde lo obtenido del análisis resultante del trabajo realizado hasta el posible trabajo a futuro.

5.1. Sobre el Trabajo Realizado

En esta memoria se decidió estudiar sobre el Análisis de Conceptos Formales, en particular, el desempeño de sus algoritmos de calculo de conceptos formales, tanto secuenciales como paralelos, intentando concluir el por que de los resultados obtenido respecto a las ejecuciones de los diversos algoritmos implementados.

Respecto a lo anterior, se puede concluir que el enfoque de paralelización de tareas es algo latente debido a la gran cantidad de datos sobre la que se esta trabajando hoy en día en relación al mundo Big Data. Es por esto, que el estudio de mejoras que permitan obtener mejores desempeños es algo crucial. Junto a esto, la implementación de algoritmos que puedan hacer uso de tecnologías de almacenamiento de datos distribuidos como MapReduce o Hadoop es algo que se volverá más necesario a medida que aumenta la cantidad de datos sobre los cuales se trabaja.

Por otro lado, se debe mencionar que la ventaja algorítmica de NextClosure sobre el resto de los algoritmos es algo destacable. Pese a ser un algoritmo secuencial, la capacidad de hacer uso del test canónico le da una riqueza algorítmica que supera incluso la paralelización del resto de los algoritmos.

Como era necesario la implementación de estos algoritmos en la creación de la librería utilizada en este estudio, se puede mencionar que el proceso de desarrollo permitió un mejor entendimiento de la lógica algorítmica que se emplea en el Análisis Formal de Conceptos, en particular, en la forma en la que se recorre el espacio de búsqueda, de manera que una buena decisión algorítmica puede ser más potente que una ventaja de hardware con paralelización.

5.2. Sobre los Resultados Obtenidos

Como se a descrito a lo largo del documento, las técnicas de Formal Concept Analysis que se utilizan para crear el retículo que contiene los conceptos formales de la *lattice*, pueden tener largos tiempos de ejecución a medida que se ejecutan bajo contextos formales donde el número de objetos o de atributos es alto, comparado con otras metodologías de clasificación como las máquinas de aprendizaje.

En base a los resultados obtenidos y el análisis realizado, podemos notar que la diferencia entre los enfoques *breadth first* y *depth first* radica en la importancia que tiene el test canónico al momento de permitirle al algoritmo NextClosure evitar calcular clausuras sobre combinaciones que, gracias al mismo test, sabe que no son un concepto formal. Esto se puede concluir gracias a que la paralelización del algoritmo Apriori no pudo lograr mejores resultados que el resto de los algoritmos, lo que nos permite concluir la relevancia e importancia del test canónico a la hora de optimizar el trabajo del cálculo de conceptos formales.

Junto con lo anterior, podemos concluir que PPPL, si bien logra incluir tanto la naturaleza de ambos algoritmos (Apriori y NextClosure), este algoritmo posee falencias respecto a lo que es la distribución balanceada de trabajo y la elección de un tamaño de carga óptimo para su ejecución. De este último punto, podemos concluir que estudiar sobre la existencia de una relación entre la naturaleza del *dataset* y el tamaño de la carga podría ser interesante de estudiar a partir de lo expuesto en la discusión.

Además, podemos concluir que la paralelización en los algoritmos trae beneficios a en términos de tiempo gracias a la paralelización de trabajo, lo que podemos ver comparando los tiempos de ejecución de Apriori y Apriori paralelo. Sin embargo, esta ganancia en tiempo por sí sola en el enfoque *breadth-first* no logra ser tan grande como la optimización durante la ejecución que brinda el test canónico. Por otro lado, si bien PPPL logra mezclar ambos enfoques, el desbalance en las cargas y el no aprovechamiento de los tiempos que los procesos quedan sin trabajo, genera que no logre tener un mejor resultado que el algoritmo NextClosure, de manera que sea más eficiente el diseño algorítmico del test canónico que la paralelización de este.

También se puede destacar que la paralelización del enfoque *depth first* en SPPL consiguió mejores resultados que la paralelización de todos los algoritmos con enfoque *breadth-first*. Además, los resultados son muy cercanos a los valores obtenidos por NextClosure, por lo que motiva al estudio en detalle de las posibles optimizaciones de dicha propuesta para obtener mejores resultados que NextClosure.

Por último, aunque se obtuvo que Apriori logró un mejor desempeño que NextClosure en los escenarios Servo y Adult Invertido, los registros de Apriori no logra superar el desempeño de NextClosure en Servo Invertido y Adult. De esta manera, pese a lo llamativo este hallazgo, el desempeño de NextClosure sigue siendo mejor que Apriori.

5.3. Sobre el Cumplimiento de los Objetivos

A continuación, se revisará el cumplimiento de los objetivos, evaluando primeramente los específicos para luego llegar al general.

Respecto al primer objetivo específico planteado que es “recolectar y analizar algoritmos existentes para el Análisis Formal de Conceptos, investigando su posible paralelización”, se puede desprender que si se cumplió. Como se observa en el capítulo dos del documento, en la sección de algoritmos secuenciales y algoritmos paralelos, se estudió e investigó sobre algoritmos existentes para FCA, revisando la paralelización de algunos de ellos. Además, en el capítulo cuatro, en la sección de implementación de los algoritmos, se estudió e implemento la paralelización del algoritmo Apriori y la implementación de PPPL y SPPPL.

En segundo lugar, el siguiente objetivo específico “diseñar e implementar una librería que esté compuesta por una selección de algoritmos de Análisis Formal de Conceptos que se puedan paralelizar”, se puede deducir que también se cumplió. Esto lo podemos evidenciar en el capítulo tres, la sección de diseño de la librería, y en el capítulo cuatro, en la sección de implementación de la librería. No esta demás mencionar que gracias a la implementación de los algoritmos que componen la librería, se pudo obtener los resultados de las ejecuciones de los diferentes algoritmos que se analizaron.

Como tercer objetivo específico “comparar el desempeño en tiempo de ejecución entre algoritmos secuenciales y algoritmos paralelos de FCA utilizando la librería implementada”, también fue logrado. Esto se puede notar por los resultados obtenidos en el capítulo cuatro del documento, en la sección de resultados obtenidos, como por ejemplo lo presentado en la tabla 9, y el análisis a estos resultados en la sección de análisis de resultados.

Por último, el objetivo general del trabajo es “analizar y comparar cuantitativamente algoritmos secuenciales y algoritmos paralelos de Análisis Formal de Conceptos en relación con sus tiempos de ejecución, uso de memoria y cálculos de clausuras”. En base a lo concluido por cada objetivo específico, se puede decir que si se cumplió la realización del análisis y la comparación de los diferentes tipos de algoritmos. A lo largo de este documento, se puede constatar un estudio que contextualiza en que consisten los algoritmos de FCA, junto con la implementación de los algoritmos seleccionados y el análisis de los resultados obtenidos en la ejecución de los *datasets* seleccionados. Además, en el capítulo cuatro, en la sección de análisis de resultados, se estudió y se planteo algunos posibles motivos que explicarían la diferencia de los resultados entre los distintos algoritmos

5.4. Trabajo a Futuro

A continuación se expondrá el trabajo a futuro que se debe realizar para poder estudiar aún más aspectos del desarrollo del algoritmos paralelos para el Análisis Formal de Conceptos.

- En primer lugar, una de las cosas que se puede desprender de PPPL, es el problema de la determinación del tamaño de la carga para obtener un desempeño óptimo. Como se mencionó en el análisis de los resultados, el valor óptimo de desempeño para la carga no parece ser una tarea trivial, ya que este varía dependiendo del contexto, y además para el mismo contexto puede tener un óptimo diferente dependiendo de las características del entorno donde es ejecutado (cantidad de núcleos y velocidad procesador). Por lo mismo, estudiar cuál es la naturaleza del comportamiento de la carga óptima al cambio de dichas variables puede aportar información valiosa para el algoritmo.
- Estudiar las posibles optimizaciones de SPPL debido a sus buenos resultados. Estudiar la relación existente entre variables del entorno de ejecución como la cantidad de núcleos, velocidad de la CPU, entre otras, afecta al resultado del algoritmo y optimizar en base a estas variables. Además, estudiar los resultados del algoritmo bajo umbrales de tiempo máximo de ejecución antes de su repartición de trabajo en nuevo proceso con diferentes valores o variables durante la misma ejecución del algoritmo dependiendo de características del mismo subproceso.
- Como tercer punto, a partir de lo estudiado en los algoritmos existentes, se pudo identificar que no existen muchos algoritmos que exploten la capacidad de la programación en paralelo en entornos de ejecución que incluyan el uso de los múltiples núcleos de las tarjetas gráficas. El aprovechar este avance tecnológico de bajo costo puede aportar al diseño de algoritmos paralelos brindándoles esta ventaja de hardware en relación a la cantidad de núcleos que poseen las GPU de hoy en día.
- En último lugar, el hallazgo respecto al comportamiento de Apriori y NextClosure es uno de los puntos relevantes encontrados en este trabajo. Estudiar respecto a que técnicas podrían permitir identificar que Apriori tendrá este tipo de comportamiento, intentar optimizar el desempeño de Apriori para intentar superar el NextClosure, o crear un algoritmo nuevo bajo estas condiciones podría ser uno de los trabajos a futuro en esta línea.

A modo de resumen, continuar con el estudio de propuestas paralelas que permitan aprovechar las capacidades de cómputo, las arquitecturas de múltiples núcleos o arquitecturas cloud, son trabajos de estudio que podrían sumar valor a las propuestas y algoritmos existentes presentados en este documento.

5.5. Palabras Finales

La base que entrega estudiar Ingeniería Civil Informática en la Universidad Técnica Federico Santa María, en relación a los conocimientos básicos del mundo de la informática sumado a la opción de que el estudiante pueda tomar cursos de especialización de su área de interés es el pilar fundamental de la elaboración de esta memoria. Los conocimientos brindados en

las asignaturas electivas de Descubrimiento de Conocimiento y Análisis Formal de Conceptos, Computación de Alto Desempeño, y Sistemas Distribuidos, fueron los que permitieron la contextualización de la complejidad algorítmica del problema a tratar, junto con la implementación de los algoritmos con los que se trabajó.

Sumado a lo anterior, los ramos como Algoritmos y Complejidad, Estructuras Discretas, Computación Distribuida para Big Data, e Inteligencia Artificial, permitieron comprender desde los puntos de vista del diseño algorítmico, la resolución de problemas desde la comprensión de la naturaleza del espacio de búsqueda, y la existencia de las tecnologías distribuidas el problema expuesto en este documento. Además, de manera no menos importante, la capacidad de ser autodidacta y el gusto por la investigación que brinda el cursar esta carrera son la base que impulsaron este trabajo de memoria.

ANEXOS

5.1. Algoritmo PPPL

5.1.1. Resultados Dataset Adult

Los resultados para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Adult son los siguientes:

Tabla 16: Resultados Algoritmo PPPL Dataset Adult.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	3.2522[s]	16944	90.5156[Mb]
8	3.4667[s]	17664	93.1445[Mb]
12	11.6124[s]	26624	97.4296[Mb]
15	50.9355 [s]	104076	127.6992[Mb]

Los gráficos resultantes son los siguientes:

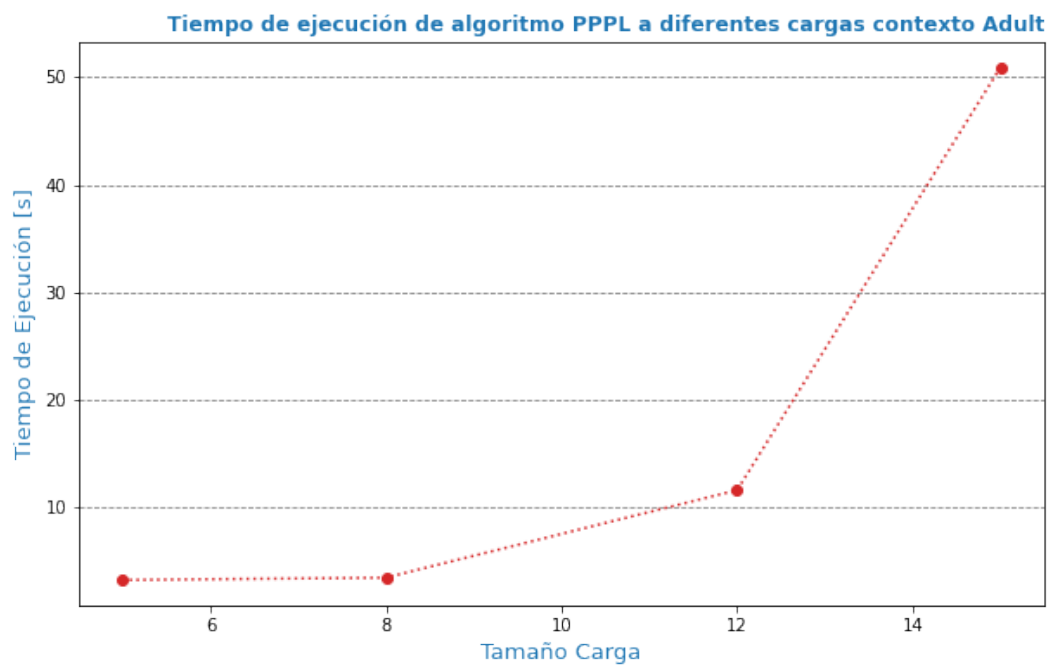


Figura 24: Resultados de Tiempo PPPL Adult

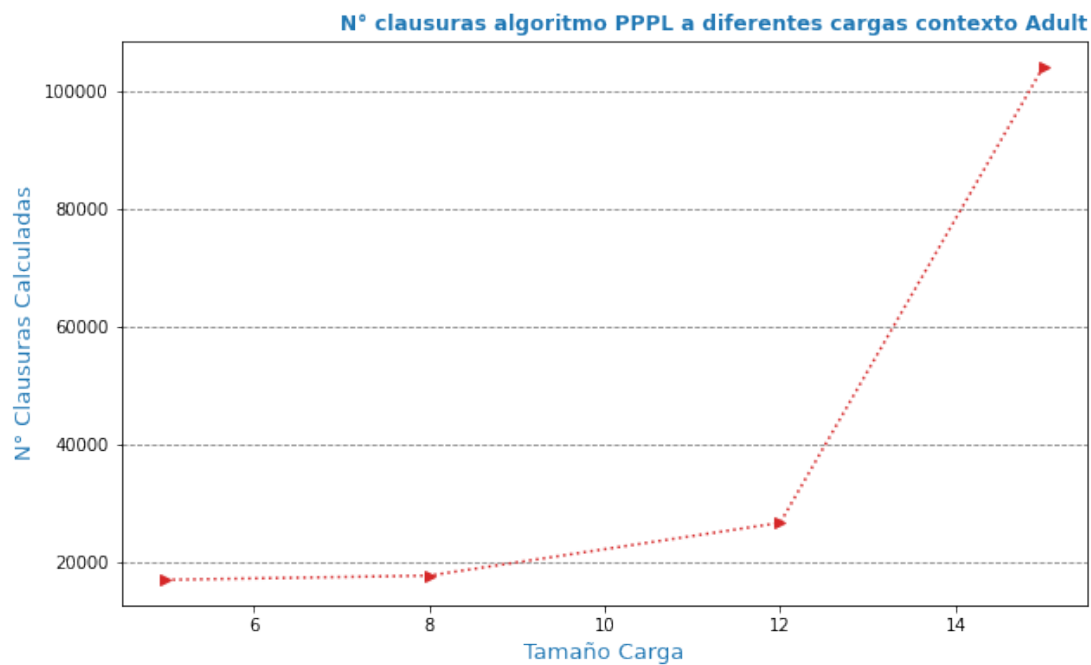


Figura 25: Resultados Número de Clausuras Calculadas PPL Adult

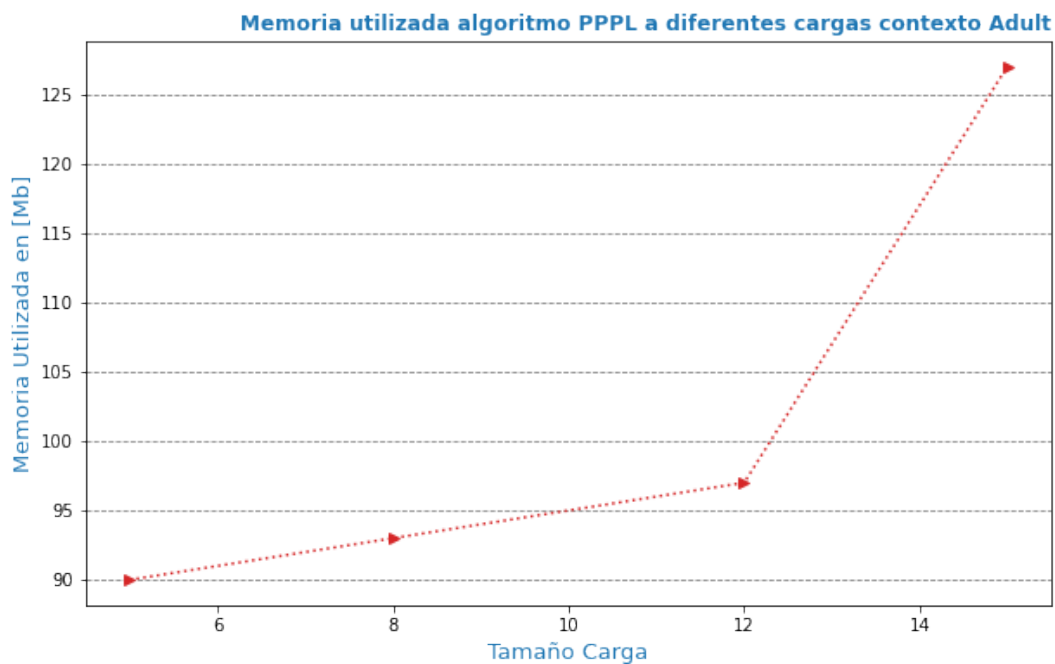


Figura 26: Resultados Memoria Utilizada PPPL Adult

5.1.2. Resultados Dataset Car

Los resultados obtenidos para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Car son los siguientes:

Tabla 17: Resultados Algoritmo PPPL Dataset Car.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	184.7123[s]	176	1455.8789[Mb]
8	343.3422[s]	448	1457.9765[Mb]
12	361.2950[s]	448	1457.8906[Mb]
15	359.1791 [s]	448	1458.1523[Mb]

Los gráficos resultantes son los siguientes:

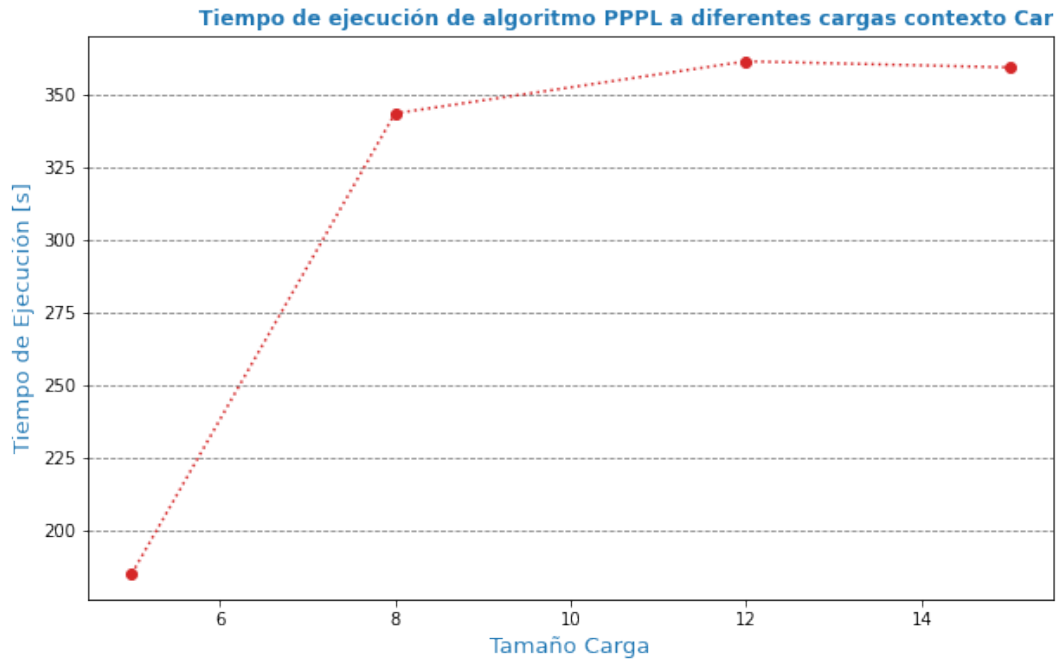


Figura 27: Resultados de Tiempo PPPL Car

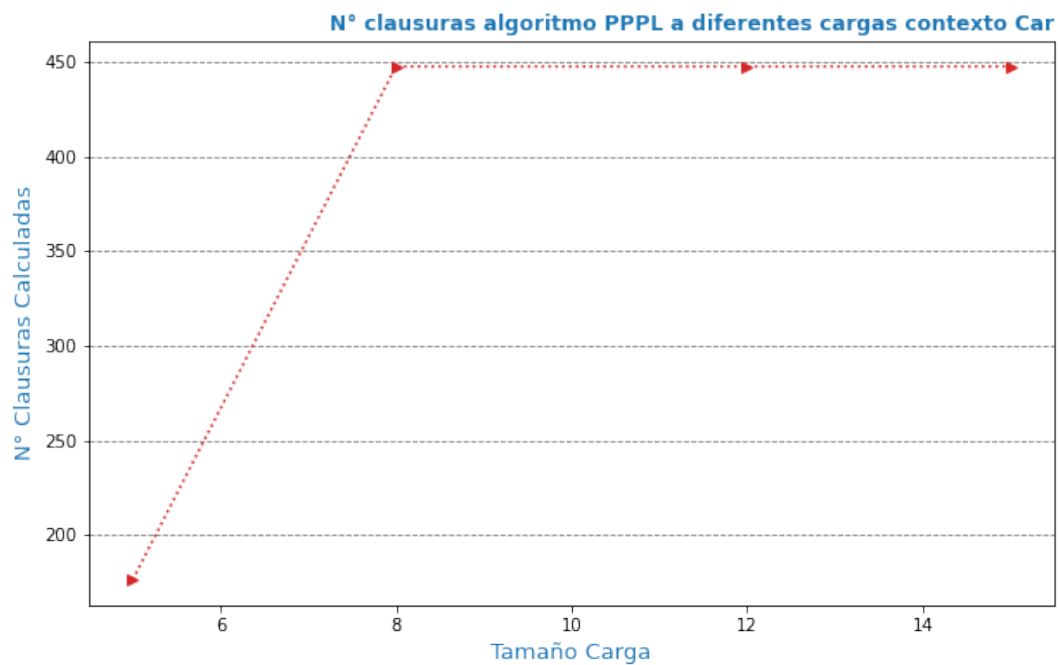


Figura 28: Resultados Número de Clausuras Calculadas PPPL Car

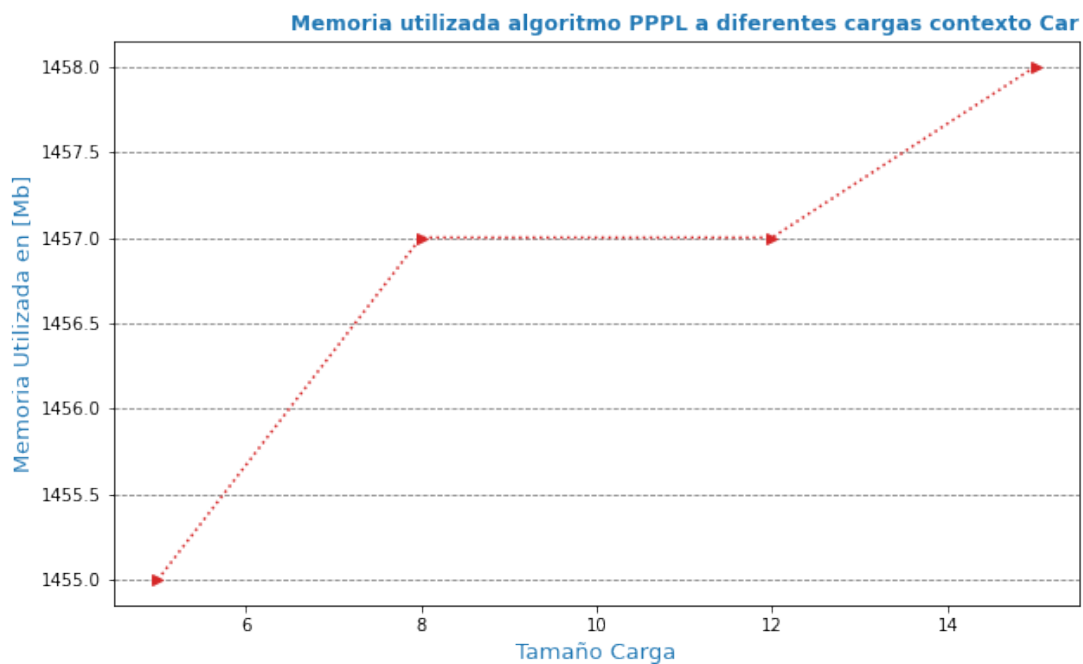


Figura 29: Resultados Memoria Utilizada PPPL Car

5.1.3. Resultados Dataset Credit

Los resultados obtenidos para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Credit son los siguientes:

Tabla 18: Resultados Algoritmo PPPL Dataset Credit.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	76.6892[s]	37667	362.2109[Mb]
8	174.4477[s]	38387	364.0039[Mb]
12	1408.9617[s]	58041	366.3125[Mb]
15	3913.6733 [s]	119745	374.9296[Mb]

Los gráficos resultantes son los siguientes:

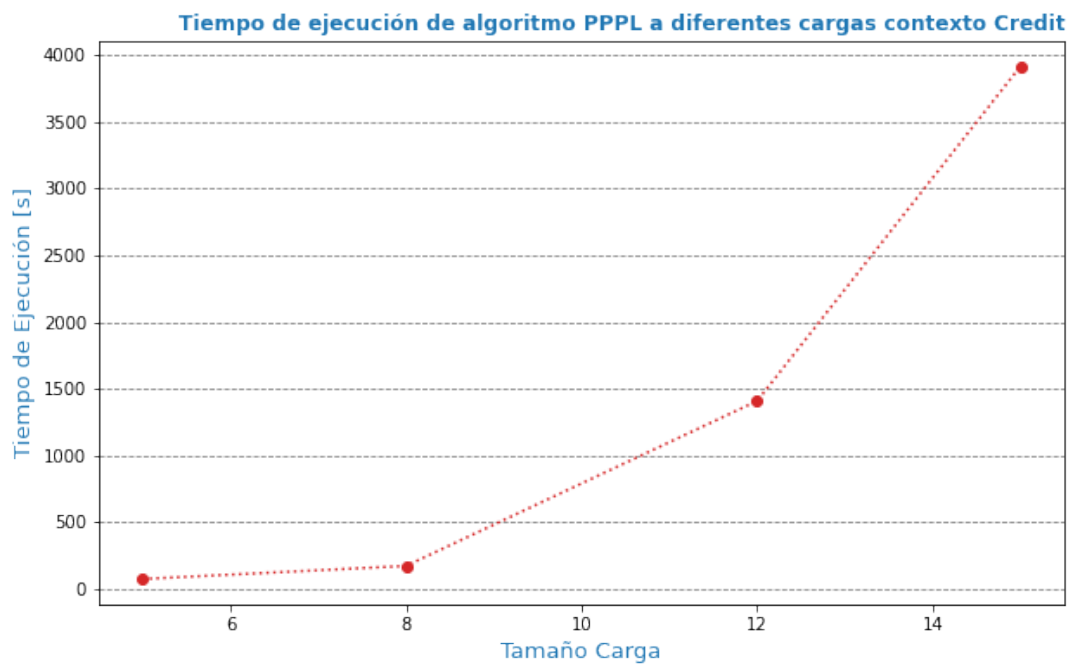


Figura 30: Resultados de Tiempo PPPL Credit

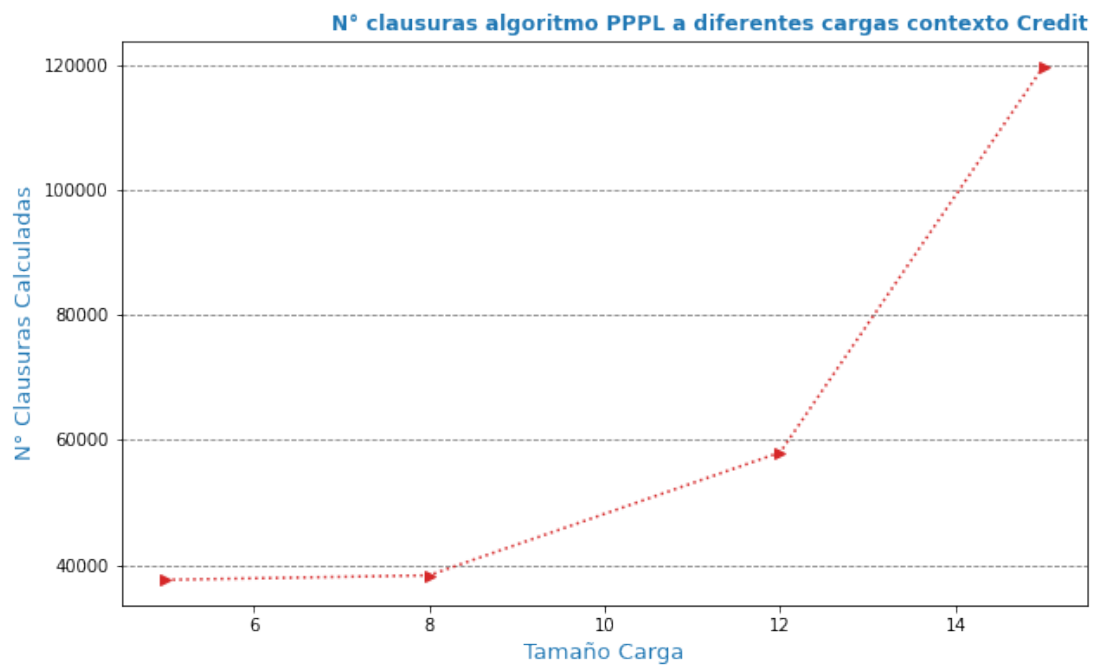


Figura 31: Resultados Número de Clausuras Calculadas PPPL Credit

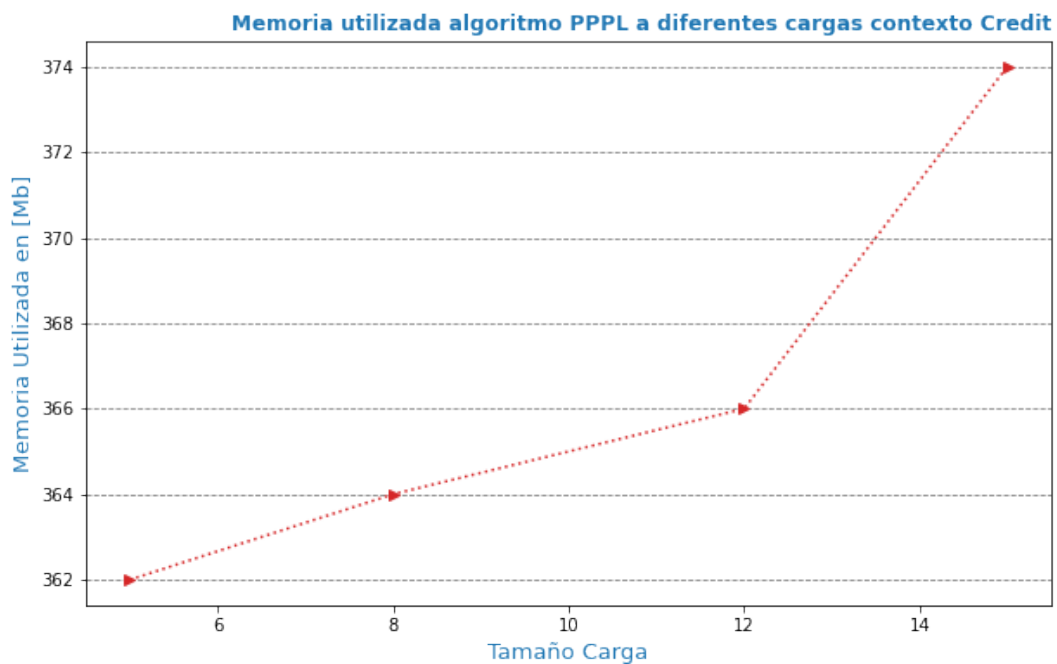


Figura 32: Resultados Memoria Utilizada PPPL Credit

5.1.4. Resultados Dataset Flight

Los resultados obtenidos para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Flight son los siguientes:

Tabla 19: Resultados Algoritmo PPPL Dataset Flight.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	2453.8300[s]	2696578	2774.8554[Mb]
8	2496.8921[s]	2696578	2775.7929[Mb]
12	2548.8044[s]	2696578	2776.0898[Mb]
15	2451.5614 [s]	2696578	2776.0117[Mb]

Los gráficos resultantes son los siguientes:

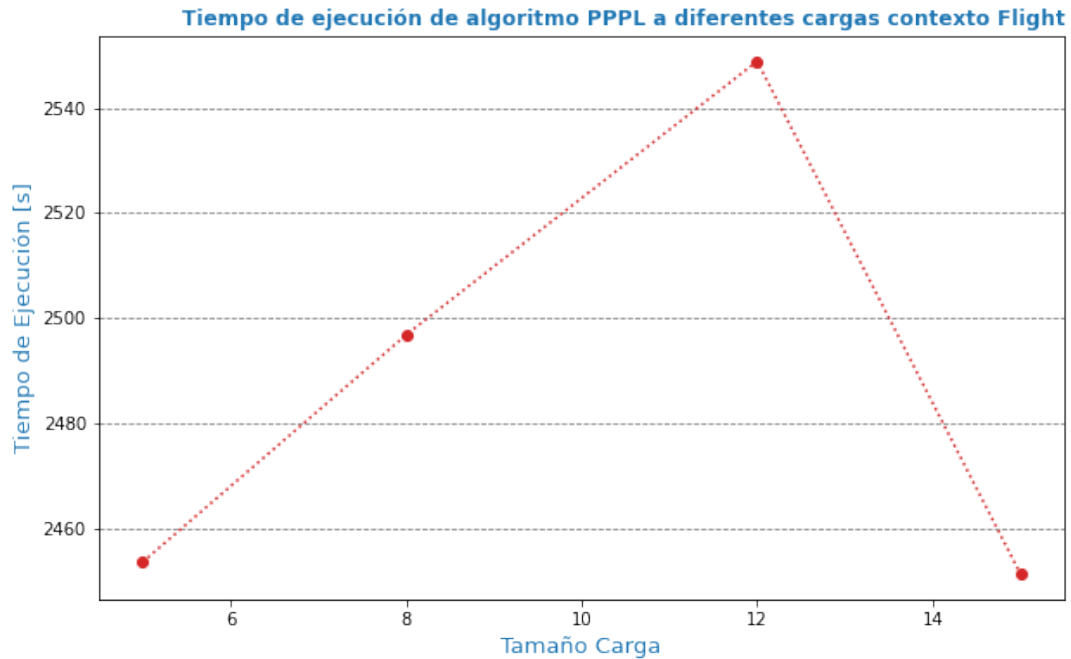


Figura 33: Resultados de Tiempo PPPL Flight

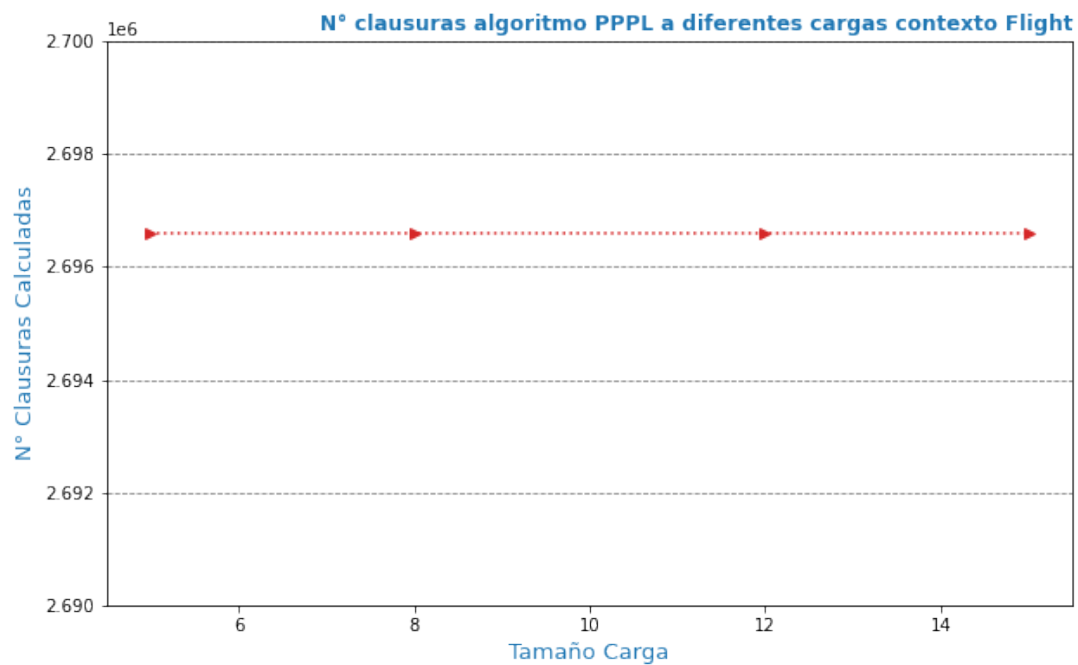


Figura 34: Resultados Número de Clausuras Calculadas PPPL Flight

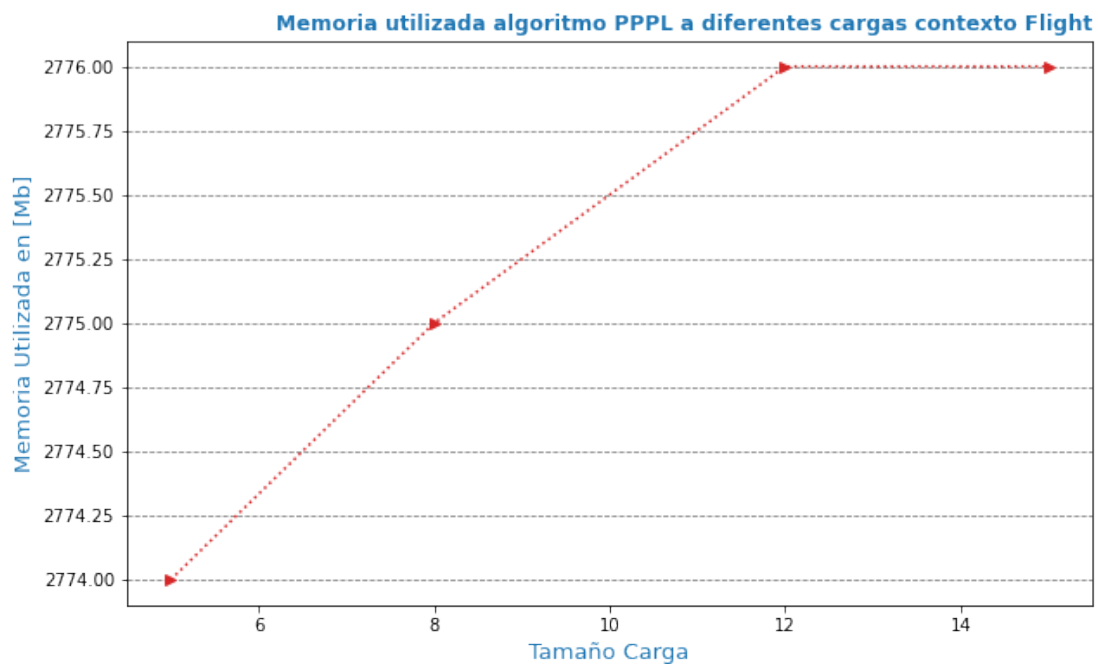


Figura 35: Resultados Memoria Utilizada PPPL Flight

5.1.5. Resultados Dataset Horse

Los resultados obtenidos para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Horse son los siguientes:

Tabla 20: Resultados Algoritmo PPPL Dataset Horse.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	2629.4565[s]	3300060	1769.5703[Mb]
8	1117.2725[s]	3300364	1031.3515[Mb]
12	1432.9785[s]	3306864	869.2265[Mb]
15	3814.5247 [s]	3337268	871.9257[Mb]

Los gráficos resultantes son los siguientes:

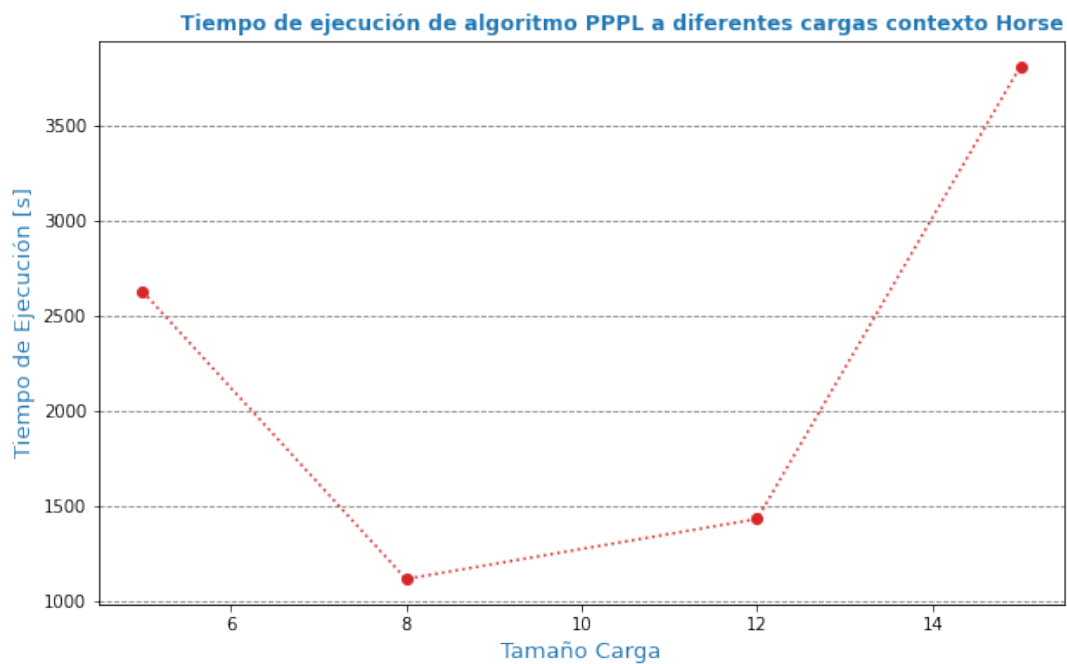


Figura 36: Resultados de Tiempo PPPL Horse

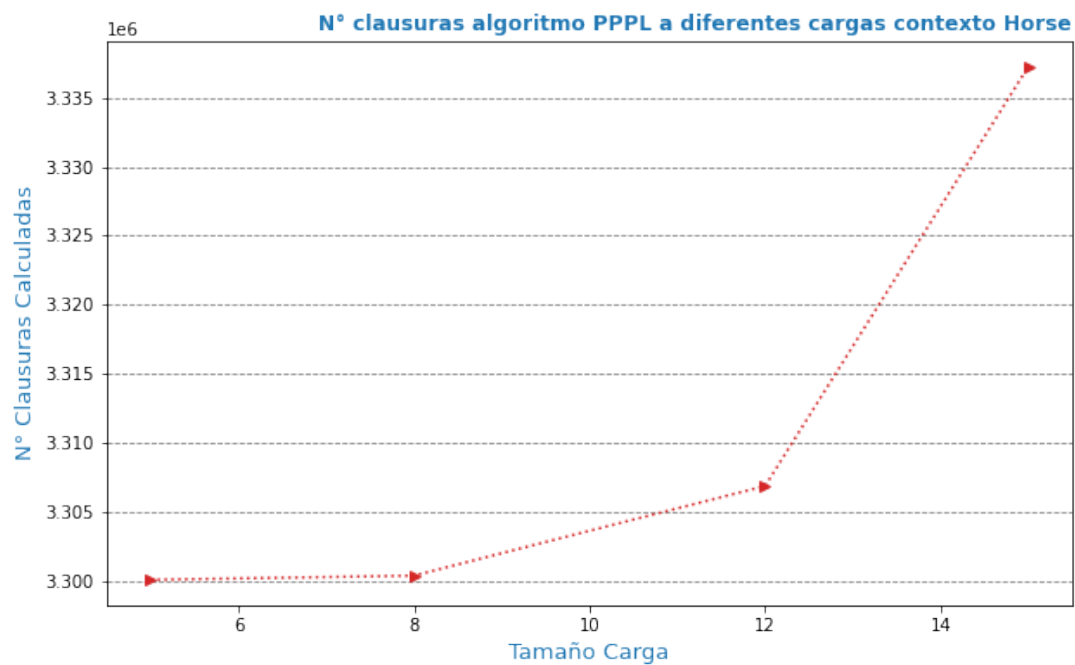


Figura 37: Resultados Número de Clausuras Calculadas PPPL Horse

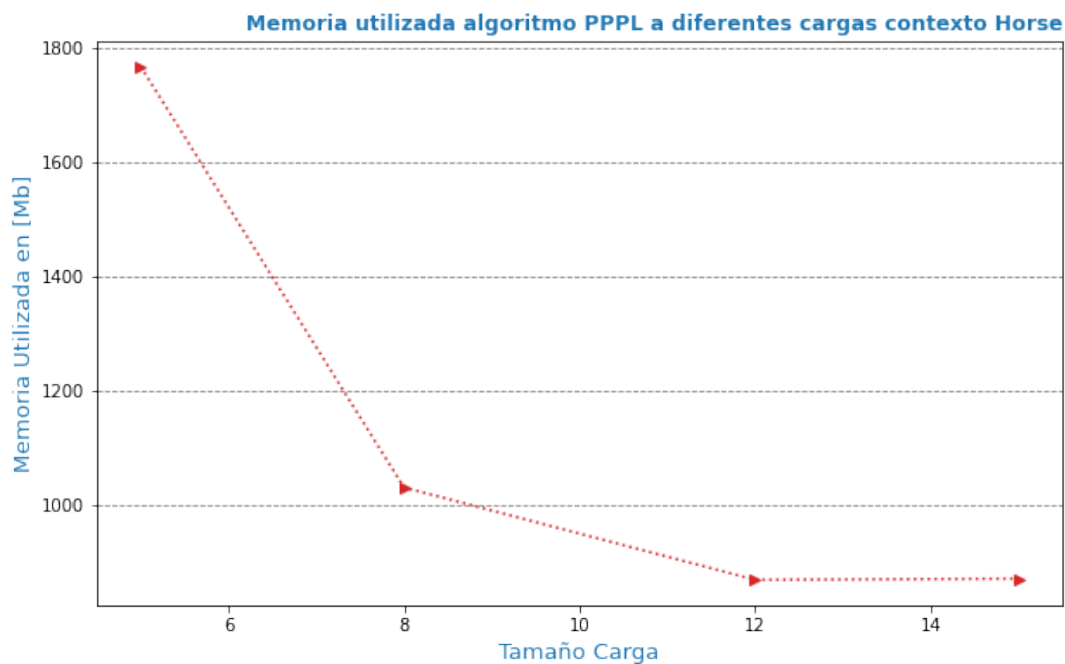


Figura 38: Resultados Memoria Utilizada PPPL Horse

5.1.6. Resultados Dataset Mushroom

Los resultados obtenidos en respecto a el tiempo de ejecución, número de clausuras calculadas y memoria utilizada para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Mushroom son los siguientes:

Tabla 21: Resultados Algoritmo PPPL Dataset Mushroom.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	337.5264[s]	1904302	793.8320[Mb]
8	103.4225[s]	1904606	694.4101[Mb]
12	81.6539[s]	1913502	700.2773[Mb]
15	277.4066 [s]	2001722	748.8945[Mb]

Los gráficos resultantes son los siguientes:

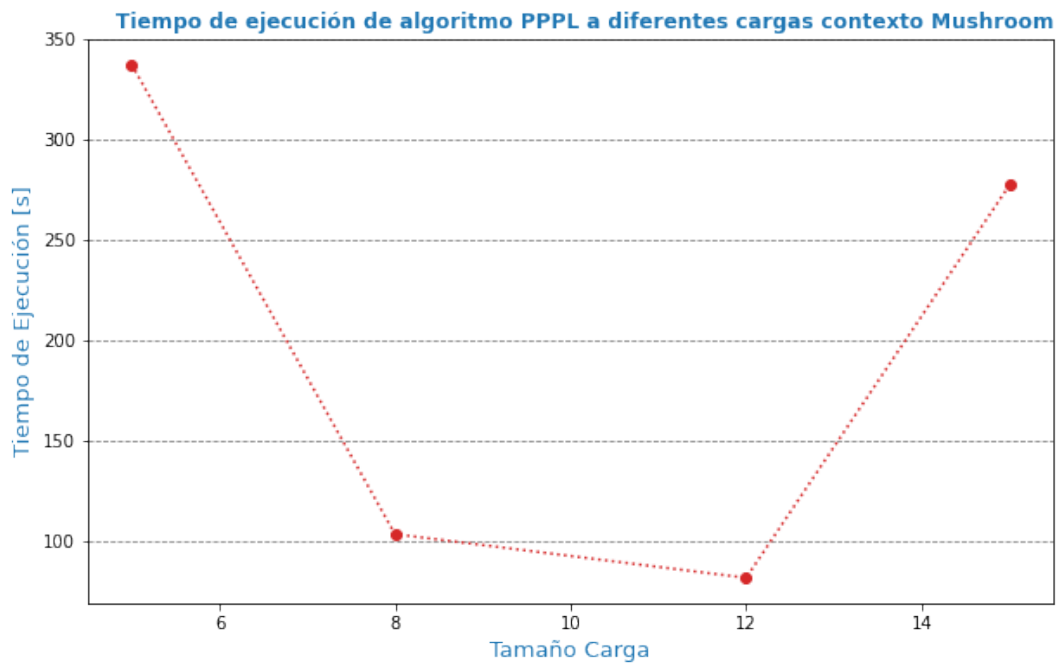


Figura 39: Resultados de Tiempo PPPL Mushroom

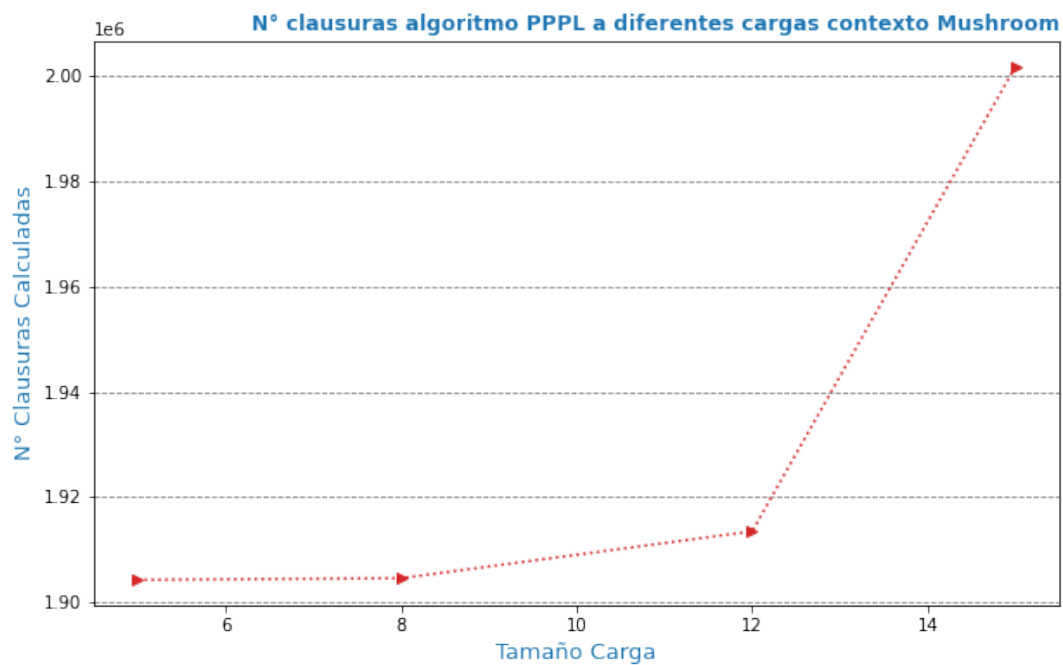


Figura 40: Resultados de Número de Clausuras Calculadas PPPL Mushroom

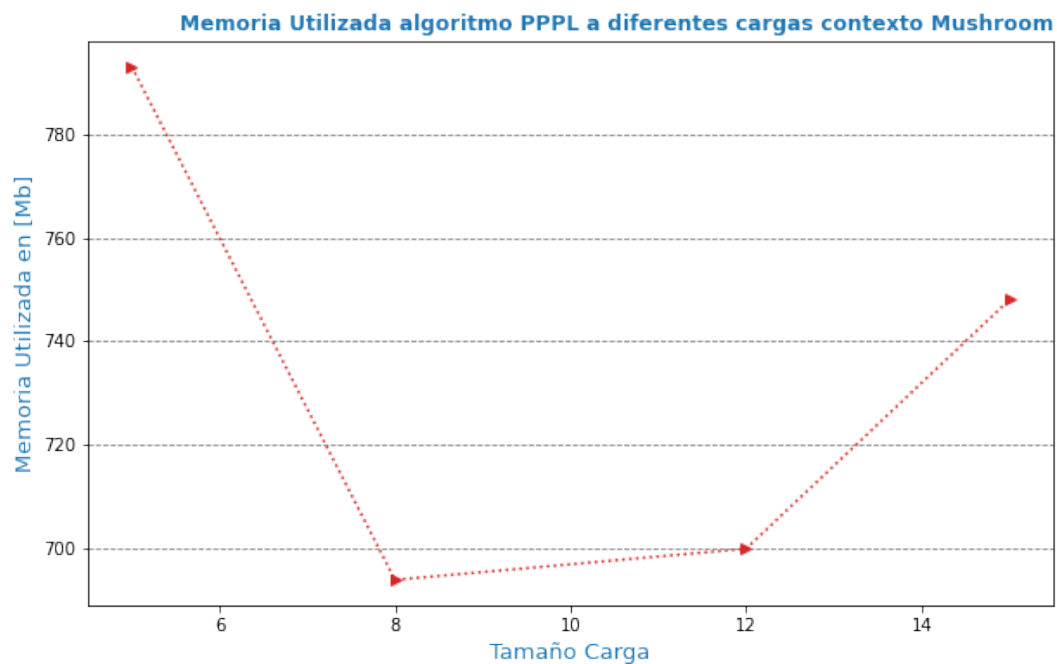


Figura 41: Resultados de uso de Memoria PPPL Mushroom

5.1.7. Resultados Dataset Servo

Los resultados obtenidos para el algoritmo PPPL con diferentes tamaños de cargas para el *dataset* Servo son los siguientes:

Tabla 22: Resultados Algoritmo PPPL Dataset Servo.

Cargas	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
5	105.3634[s]	208204	269.0976[Mb]
8	105.4451[s]	208204	268.2109[Mb]
12	104.6222[s]	208204	269.03125[Mb]
15	105.9188 [s]	208204	268.8437[Mb]

Los gráficos resultantes son los siguientes:

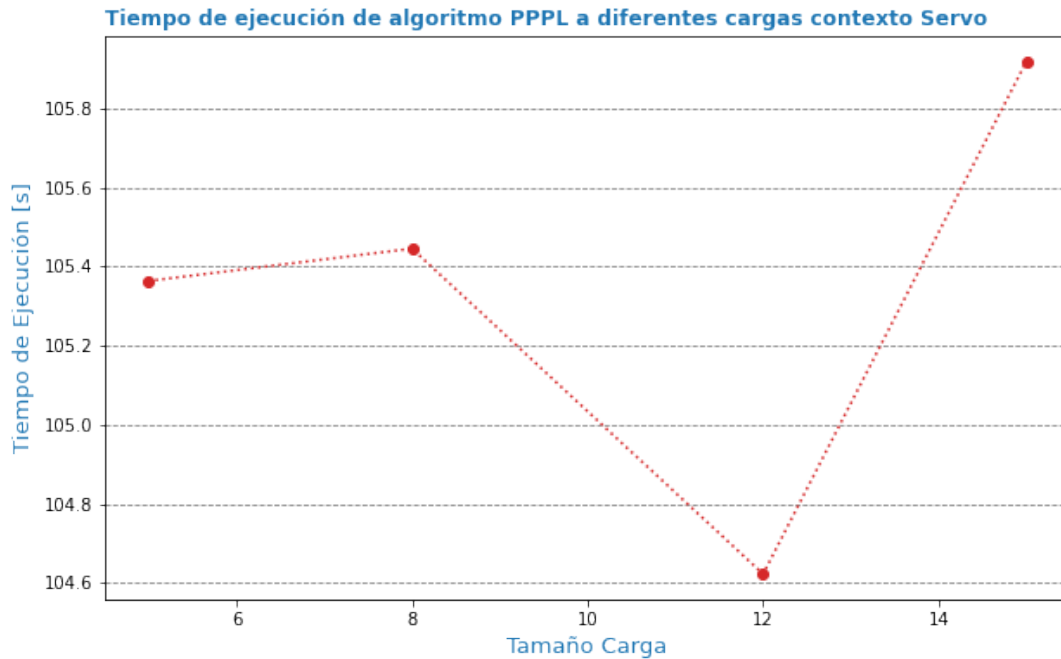


Figura 42: Resultados de Tiempo PPPL Servo

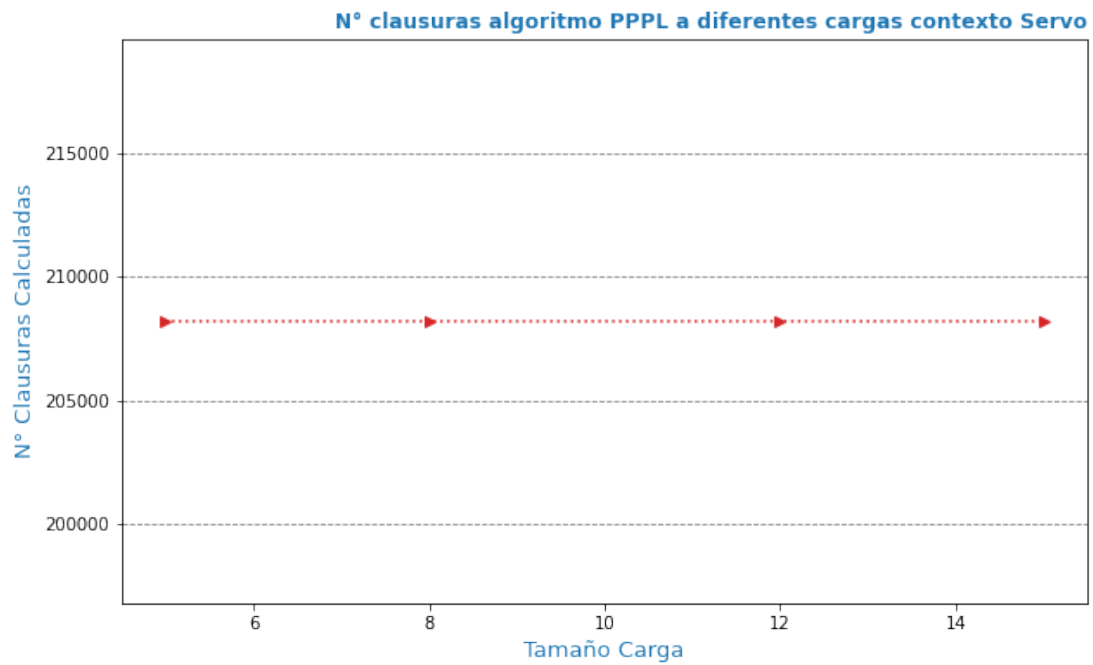


Figura 43: Resultados Número de Clausuras Calculadas PPPL Servo

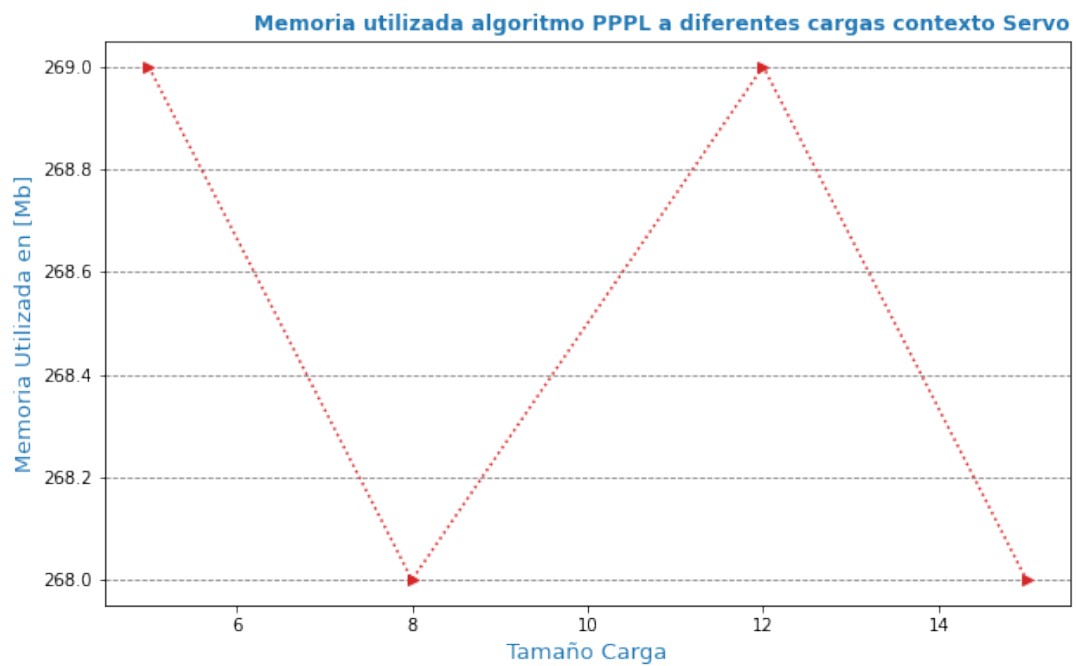


Figura 44: Resultados Memoria Utilizada PPPL Servo

5.2. Comparación General

En esta sección se presentan los gráficos en detalle para los diferentes datasets utilizados en este trabajo.

5.2.1. Dataset Adult

Los resultados obtenidos en tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Adult fueron los siguientes

Tabla 23: Resultados Algoritmos Dataset Adult.

Algoritmo	Tiempo Ejecución	Nº Clausuras	Memoria Utilizada
Apriori	8.0558[s]	200340	21.7265[Mb]
Apriori Paralelo	7.7237[s]	200340	124.6367 [Mb]
PPPL	3.2522[s]	16944	90.5156[Mb]
SPPL	1.8284[s]	16890	84.1992[Mb]
NextClosure	0.1842[s]	16896	15.3125[Mb]

Los gráficos resultantes son los siguientes:

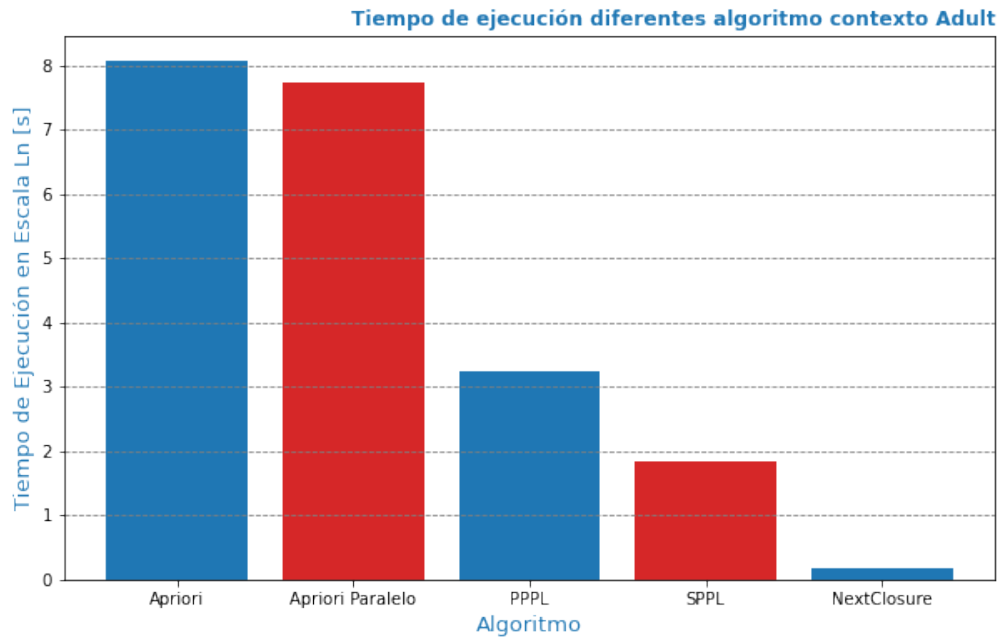


Figura 45: Resultados Tiempos Dataset Adult

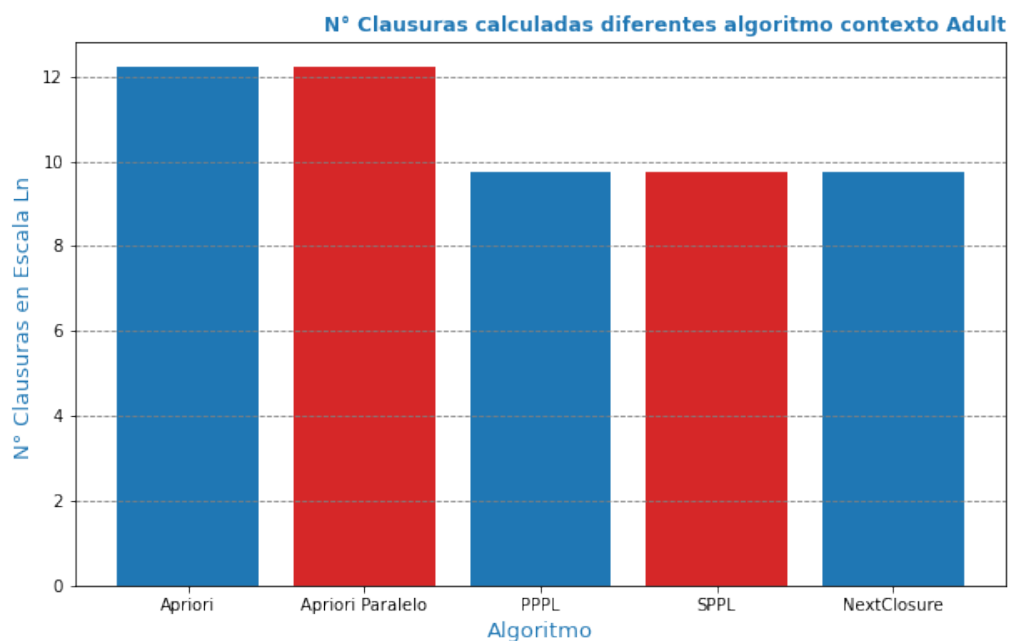


Figura 46: Resultados Número de Clausuras Calculadas Adult

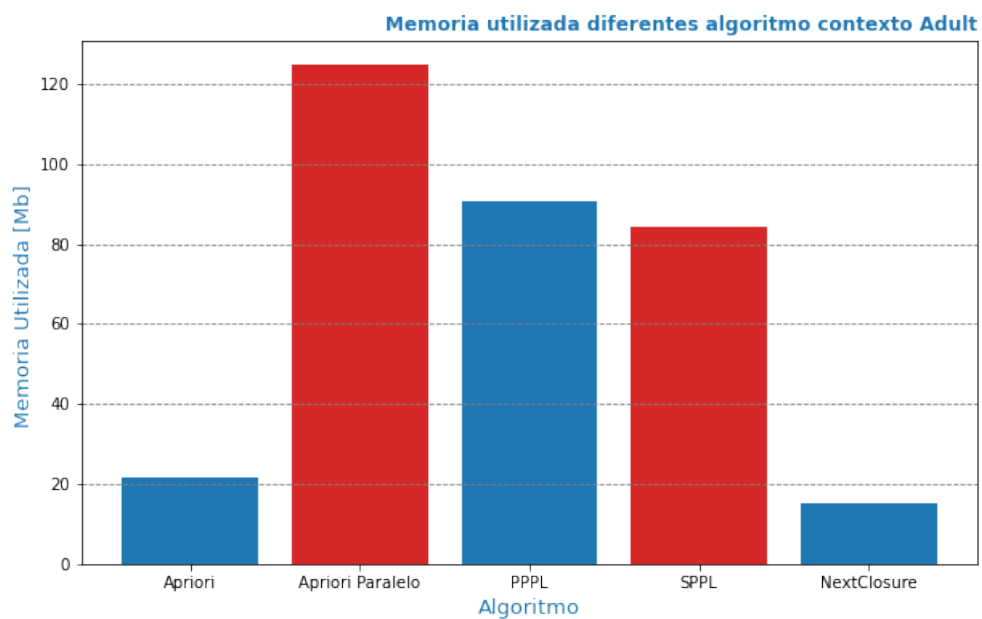


Figura 47: Resultados Memoria Utilizada Adult

5.2.2. Dataset Car

Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Car fueron los siguientes

Tabla 24: Resultados Algoritmos Dataset Car.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	113.4205[s]	889	527.9687 [Mb]
Apriori Paralelo	102.7258[s]	889	578.3320 [Mb]
PPPL	184.7123[s]	176	1455.8789[Mb]
SPPL	16.9712[s]	124	610.1835[Mb]
NextClosure	7.5078[s]	128	632.8828[Mb]

Los gráficos resultantes son los siguientes:

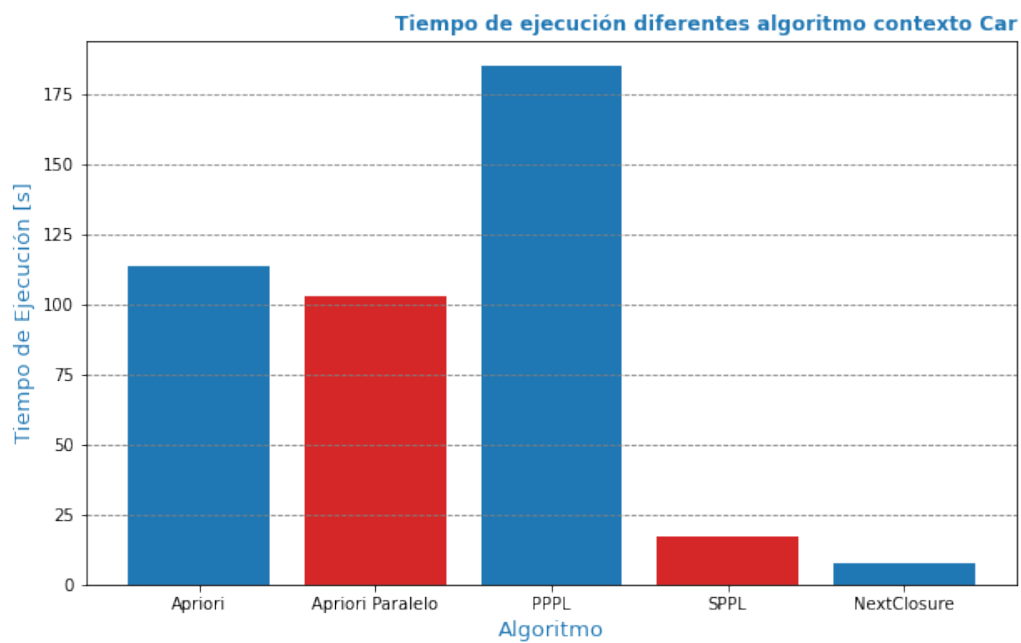


Figura 48: Resultados Tiempos Dataset Car

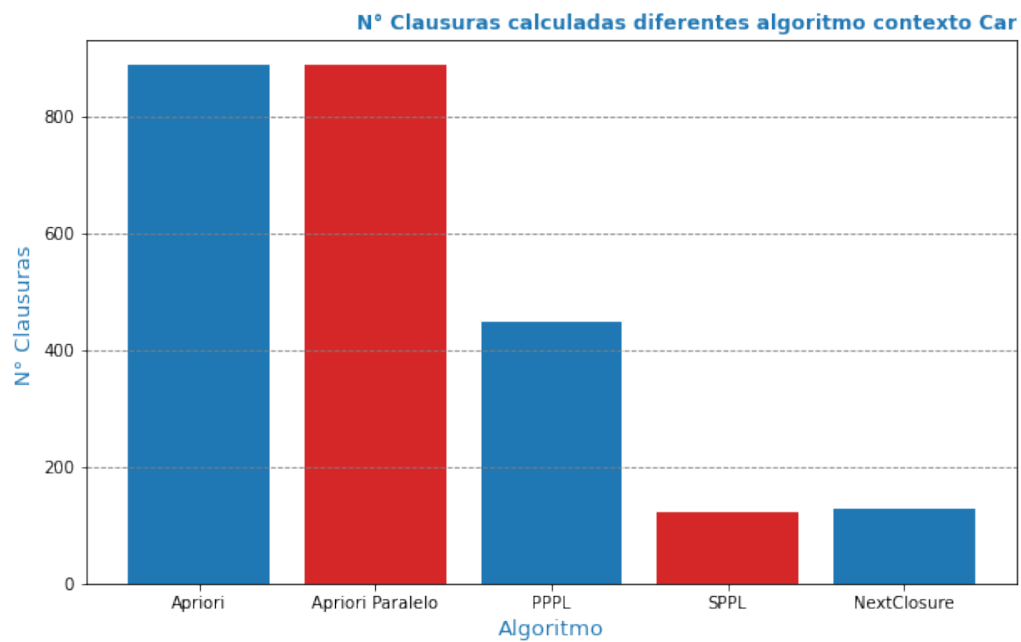


Figura 49: Resultados Número de Clausuras Calculadas Car

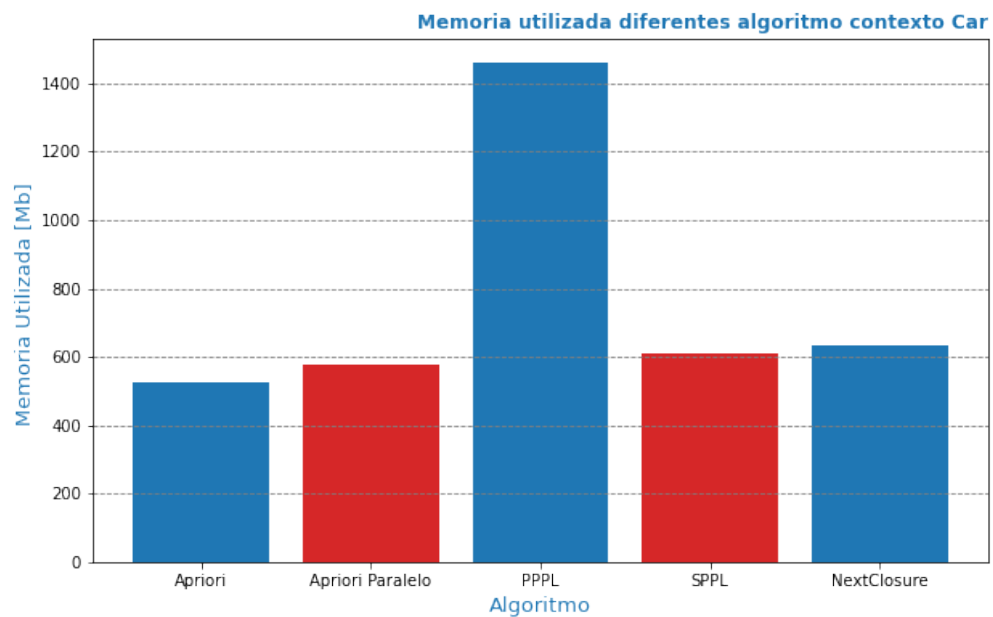


Figura 50: Resultados Memoria Utilizada Car

5.2.3. Dataset Credit

Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Credit fueron los siguientes

Tabla 25: Resultados Algoritmos Dataset Credit.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	5231.9614[s]	232944	182.7656[Mb]
Apriori Paralelo	649.3346[s]	232944	491.8515 [Mb]
PPPL	76.6892[s]	37667	362.2109[Mb]
SPPL	7.4807[s]	37608	264.8046[Mb]
NextClosure	5.3883[s]	37619	205.9335[Mb]

Los gráficos resultantes son los siguientes:

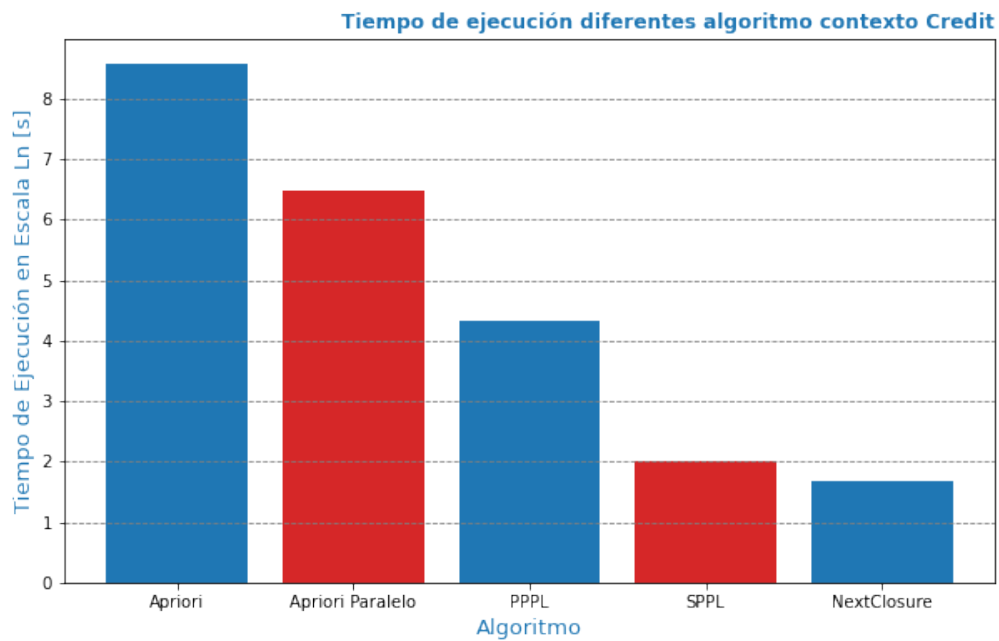


Figura 51: Resultados Tiempos Dataset Credit

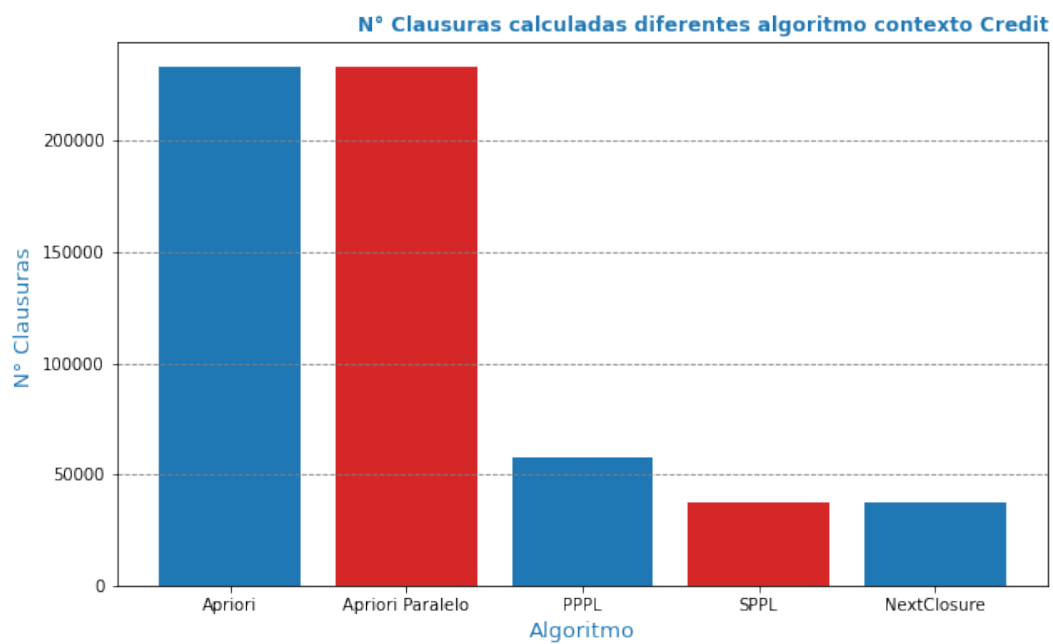


Figura 52: Resultados Número de Clausuras Calculadas Credit

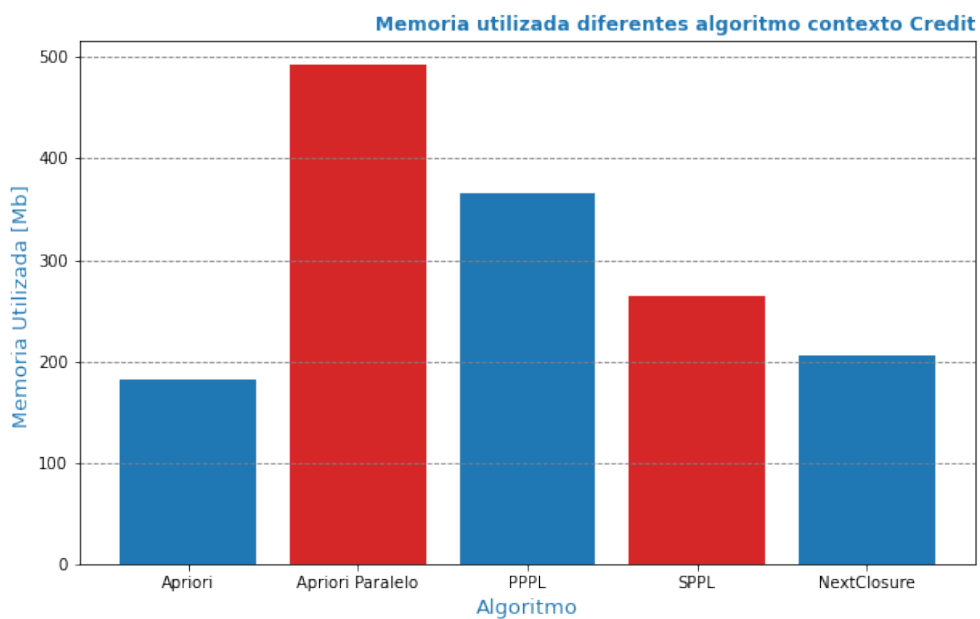


Figura 53: Resultados Memoria Utilizada Credit

5.2.4. Dataset Flight

Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Flight fueron los siguientes

Tabla 26: Resultados Algoritmos Dataset Flight.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	3559.2805[s]	21409343	788.5039[Mb]
Apriori Paralelo	912.6865[s]	21409343	2391.7890 [Mb]
PPPL	2451.5614[s]	2696578	2776.0117[Mb]
SPPL	14.4910[s]	1904282	237.4609[Mb]
NextClosure	27.5380[s]	1904286	202.875[Mb]

Los gráficos resultantes son los siguientes:

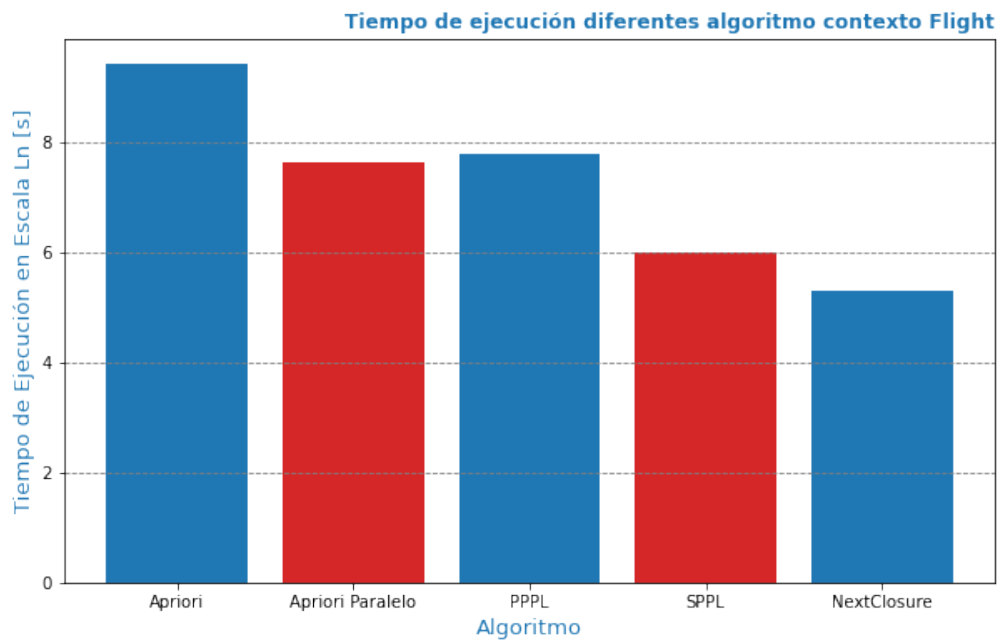


Figura 54: Resultados Tiempos Dataset Flight

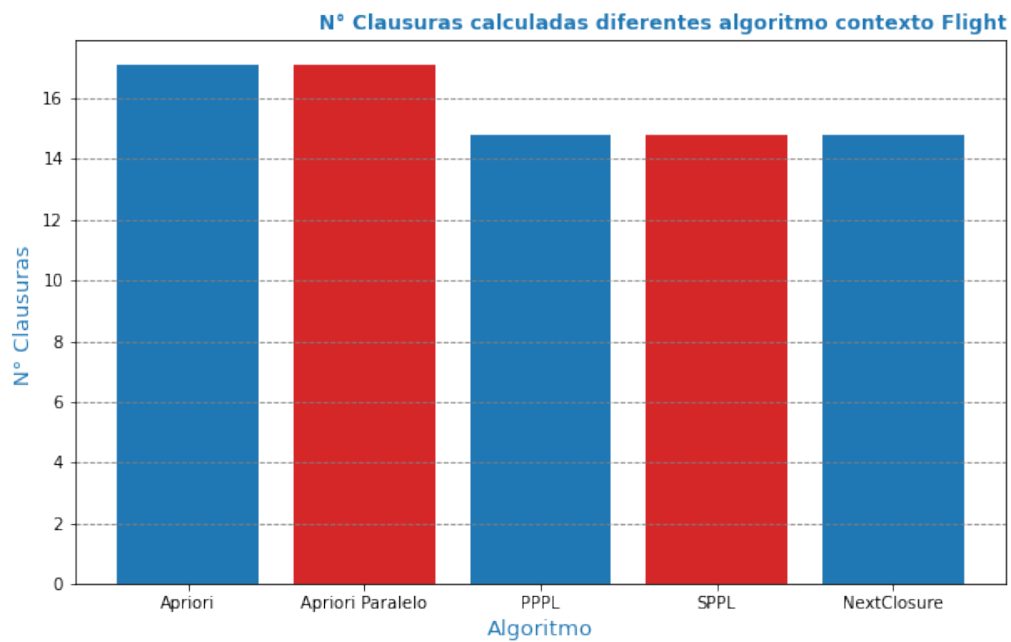


Figura 55: Resultados Número de Clausuras Calculadas Flight

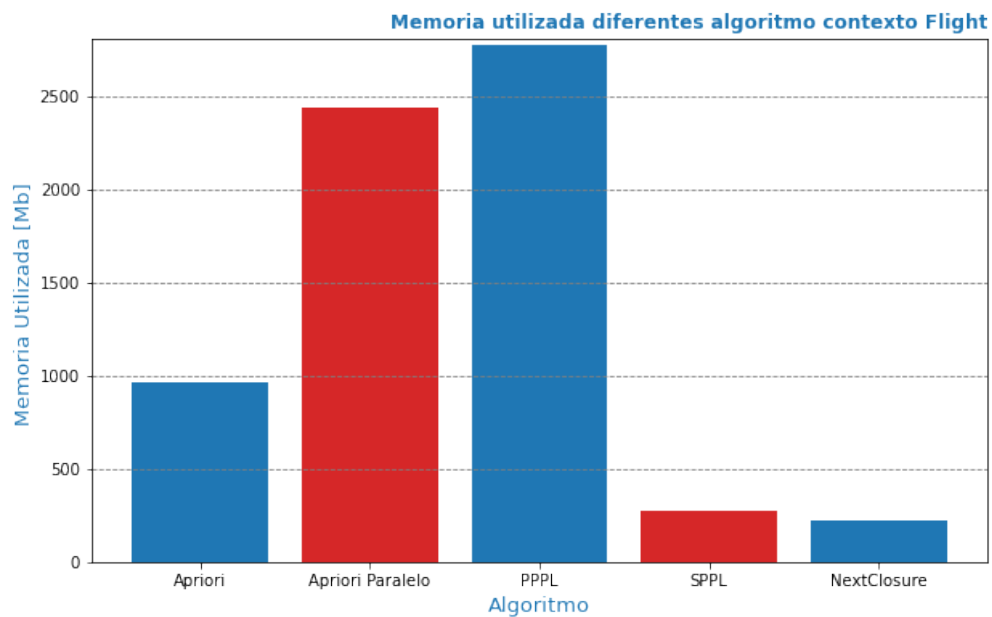


Figura 56: Resultados Memoria Utilizada Flight

5.2.5. Dataset Horse

Cabe mencionar que para este *dataset*, el algoritmo Apriori tardó más de nueve horas en su ejecución por lo que se detuvo su ejecución no rescatando todos sus datos. Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Horse fueron los siguientes

Tabla 27: Resultados Algoritmos Dataset Horse.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	34320.0000[s]	-	- [Mb]
Apriori Paralelo	14571.9325[s]	27246689	3644.5664 [Mb]
PPPL	1117.2725[s]	3300364	1031.3515[Mb]
SPPPL V2	204.5191[s]	3300006	332.4453[Mb]
NextClosure	164.3000[s]	3300044	312.0234[Mb]

Los gráficos resultantes son los siguientes:

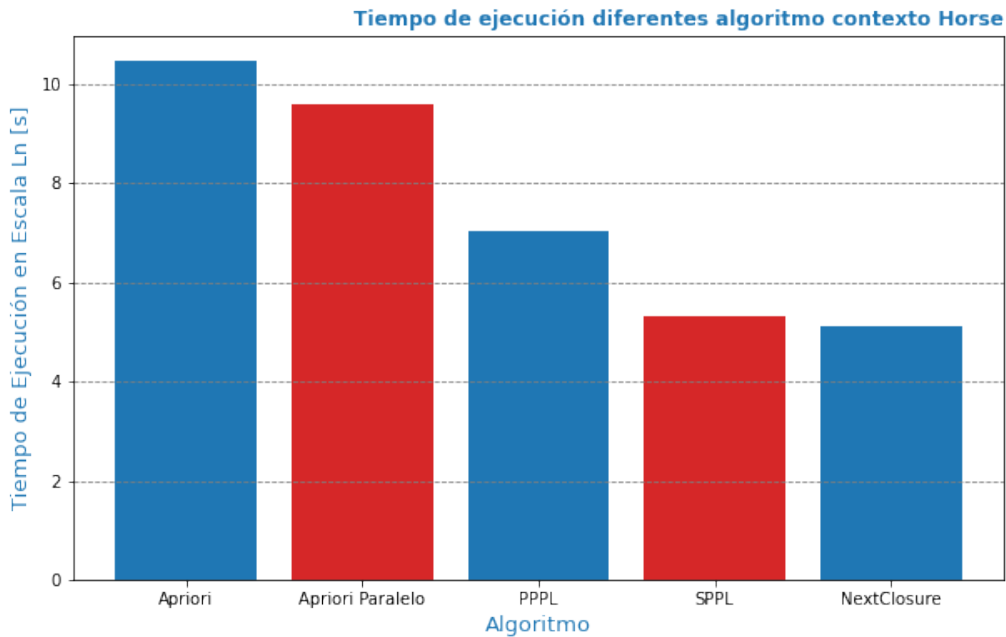


Figura 57: Resultados Tiempos Dataset Horse

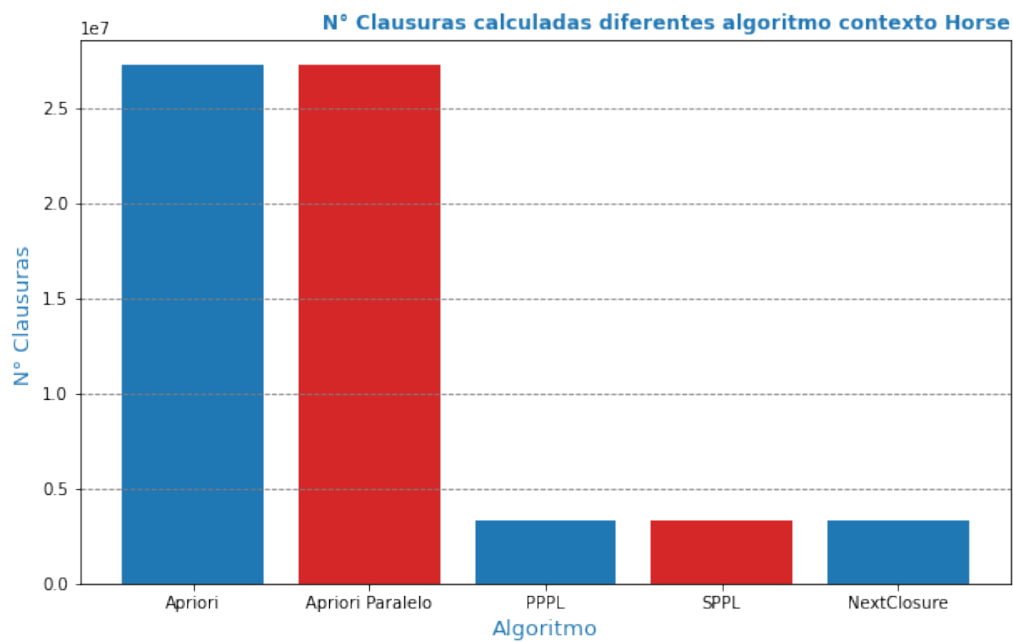


Figura 58: Resultados Número de Clausuras Calculadas Horse

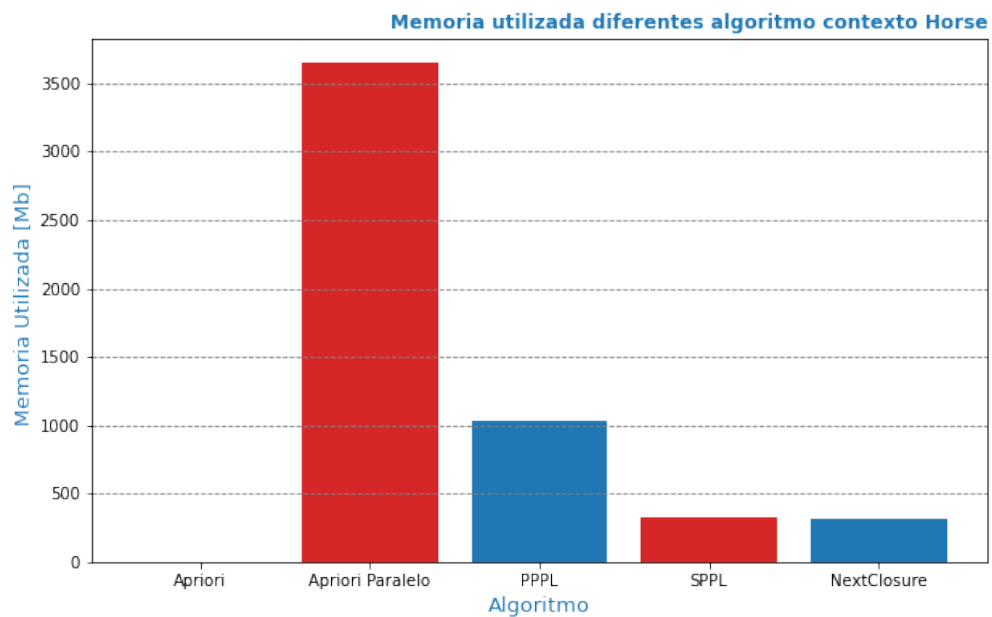


Figura 59: Resultados Memoria Utilizada Horse

5.2.6. Dataset Mushroom

Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Mushroom fueron los siguientes:

Tabla 28: Resultados Algoritmos Dataset Mushroom.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	3559.2805[s]	21409343	788.5039[Mb]
Apriori Paralelo	912.6865[s]	21409343	2391.7890 [Mb]
PPPL	81.6539[s]	1913502	700.2773[Mb]
SPPL V2	14.4910[s]	1904282	237.4609[Mb]
NextClosure	27.5380[s]	1904286	202.875[Mb]

Los gráficos resultantes son los siguientes:

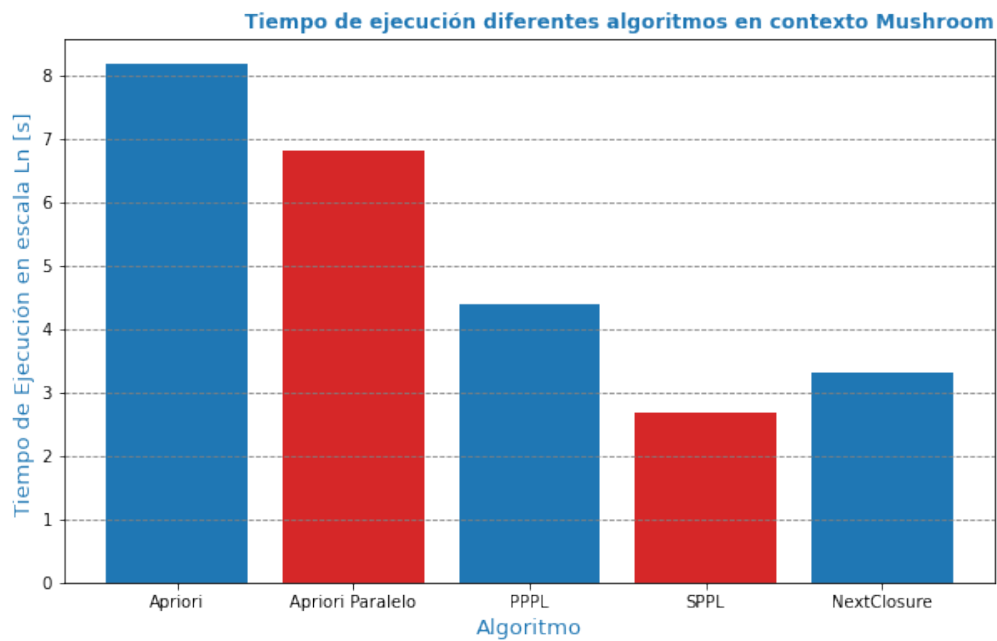


Figura 60: Resultados Tiempos Mushroom

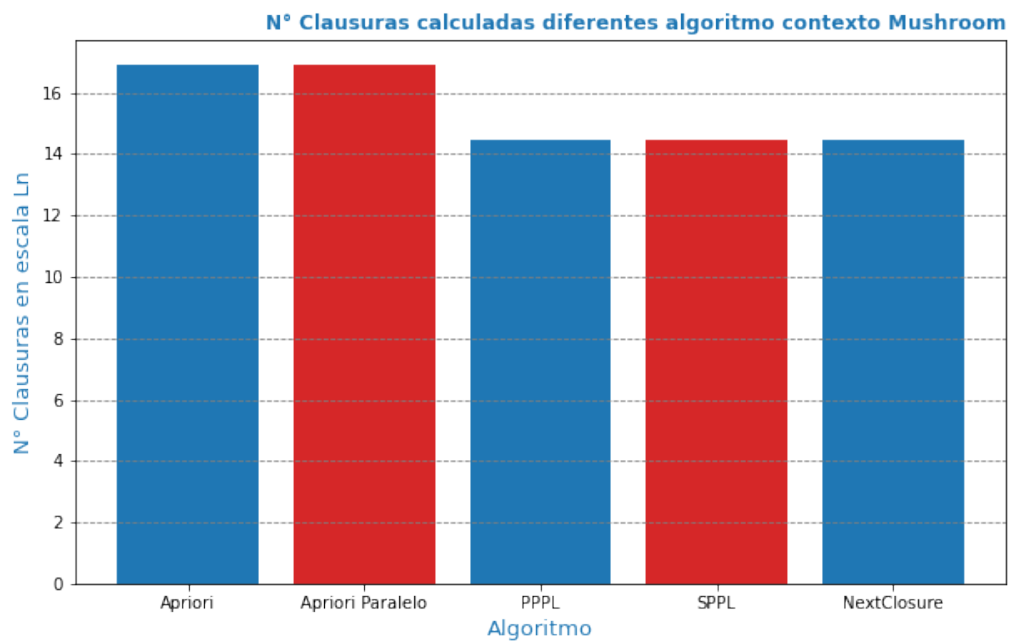


Figura 61: Resultados Número de Clausuras Calculadas Mushroom

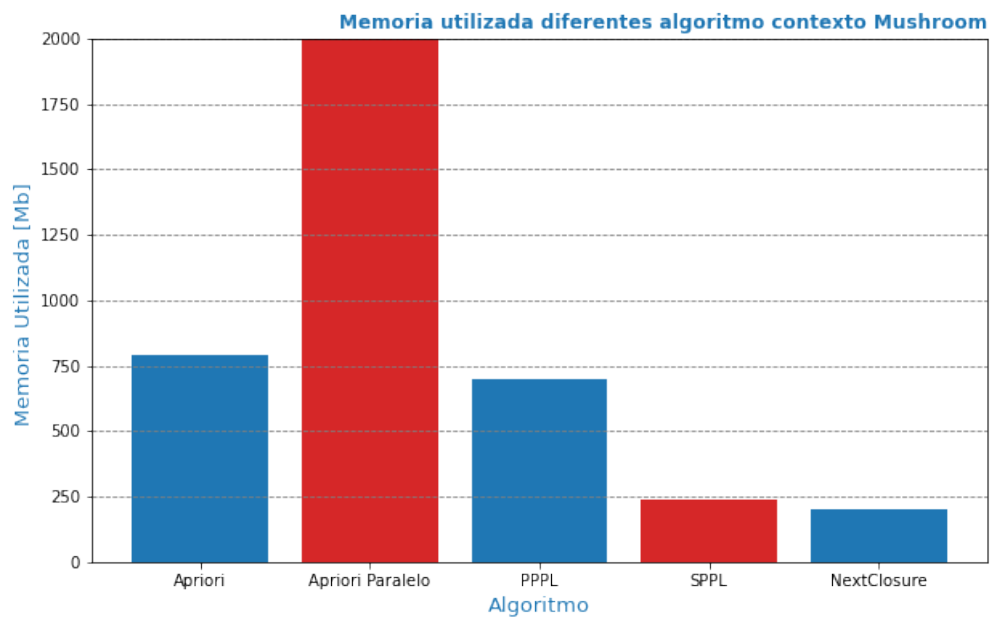


Figura 62: Resultados Memoria Utilizada Mushroom

5.2.7. Dataset Servo

Los resultados obtenidos de tiempo de ejecución, número de clausuras calculadas y memoria utilizada para todos los algoritmos en el contexto Servo fueron los siguientes

Tabla 29: Resultados Algoritmos Dataset Servo.

Algoritmo	Tiempo Ejecución	Número de Clausuras Calculadas	Memoria Utilizada
Apriori	42.3018[s]	231680	18.9648 [Mb]
Apriori Paralelo	37.0404[s]	231680	68.9179 [Mb]
PPPL	104.6222[s]	208204	269.03125[Mb]
SPPL V2	108.3769[s]	293760	87.1835[Mb]
NextClosure	172.4305[s]	208205	17.9140[Mb]

Los gráficos resultantes son los siguientes:

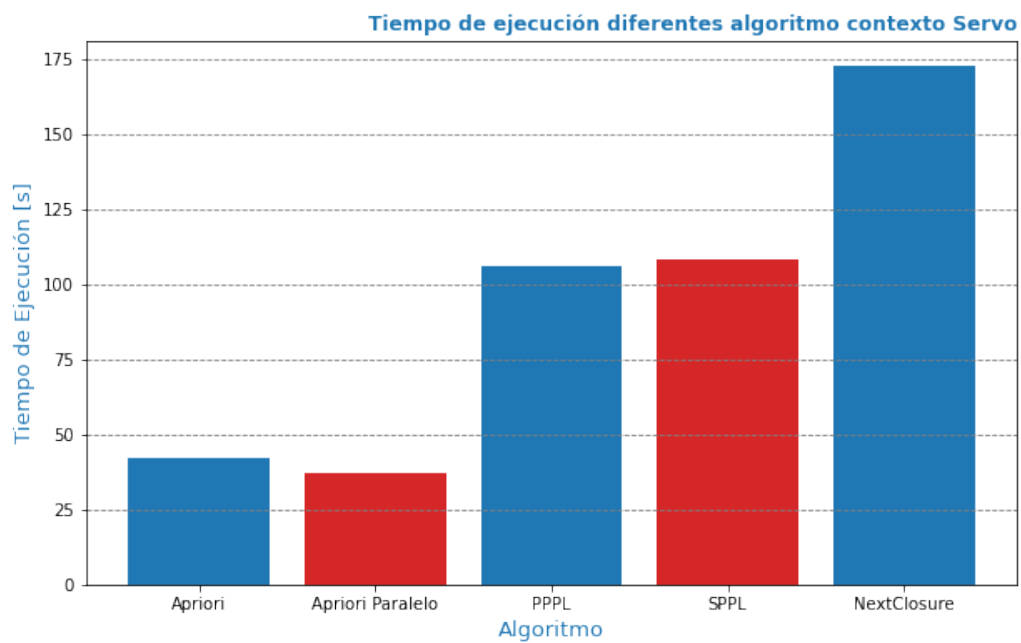


Figura 63: Resultados Tiempos Servo

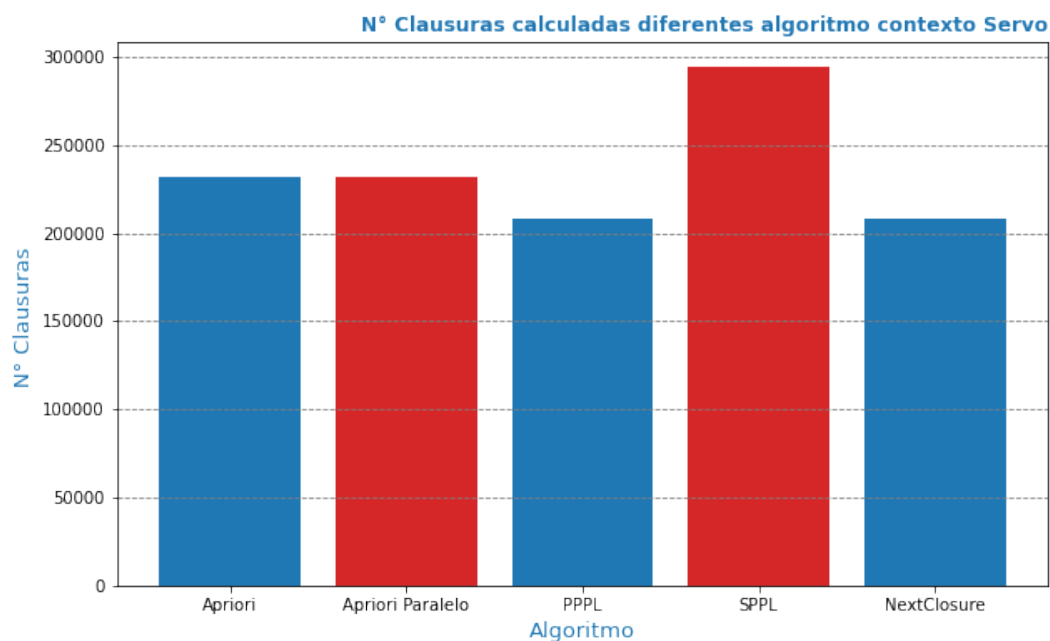


Figura 64: Resultados Número de Clausuras Calculadas Servo

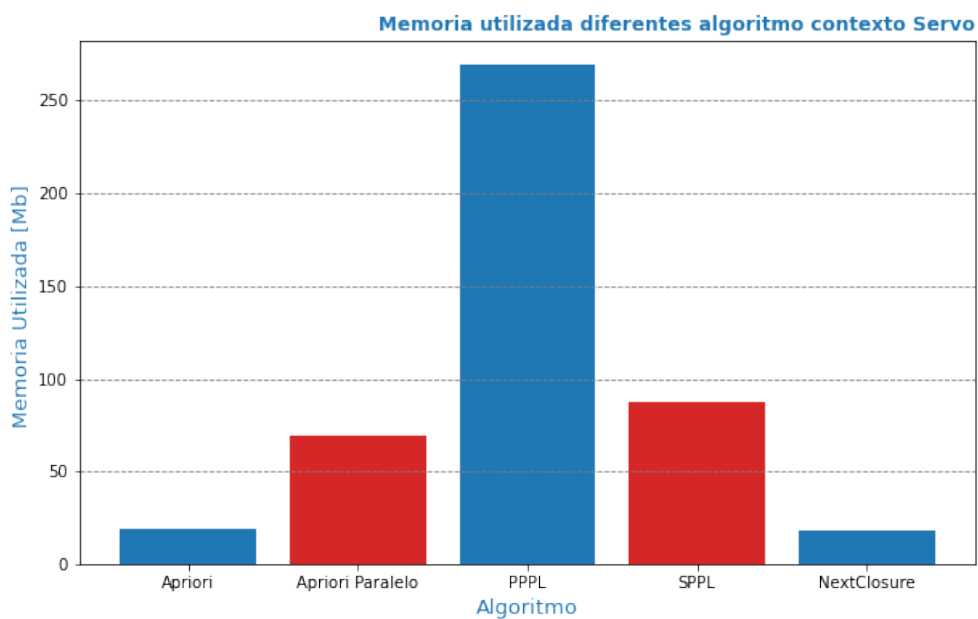


Figura 65: Resultados Memoria Utilizada Servo

REFERENCIAS BIBLIOGRÁFICAS

- [Agrawal y Srikant, 1994] Agrawal, R. y Srikant, R. (1994). Fast algorithms for mining association rules. En *Proc. 20th int. conf. very large data bases, VLDB*, volumen 1215, pp. 487–499. Citeseer.
- [Bazin y Ganascia, 2016] Bazin, A. y Ganascia, J.-G. (2016). Computing the duquenne-guigues basis: an algorithm for choosing the order. *International Journal of General Systems*, 45(2):57–85.
- [Cimiano, 2005] Cimiano, P. (2005). *Formal Concept Analysis*. Carnegie Mellon University.
- [de Moraes et al., 2016] de Moraes, N. R., Dias, S. M., Freitas, H. C., y Zárate, L. E. (2016). Parallelization of the next closure algorithm for generating the minimum set of implication rules. *Artif. Intell. Res.*, 5(2):40–54.
- [Ganter et al., 2016] Ganter, B., Obiedkov, S., Rudolph, S., y Stumme, G. (2016). *Conceptual exploration*. Springer.
- [Ganter y Wille, 1999a] Ganter, B. y Wille, R. (1999a). Contextual attribute logic. En *International Conference on Conceptual Structures*, pp. 377–388. Springer.
- [Ganter y Wille, 2012] Ganter, B. y Wille, R. (2012). *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.
- [Ganter y Wille, 1999b] Ganter, R. y Wille, R. (1999b). *Formal concept analysis: Mathematical foundations* springer-verlag berlin germany.
- [González et al., 2003] González, M. T. H., Albert, J. L., y Artalejo, M. R. (2003). *Matemática discreta y lógica matemática*. Editorial complutense.
- [Kaytoue et al., 2015] Kaytoue, M., Codocedo, V., Buzmakov, A., Baixeries, J., Kuznetsov, S. O., y Napoli, A. (2015). Pattern structures and concept lattices for data mining and knowledge processing. En *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 227–231. Springer.
- [Krajca et al., 2010] Krajca, P., Outrata, J., y Vychodil, V. (2010). Parallel algorithm for computing fixpoints of galois connections. *Annals of Mathematics and Artificial Intelligence*, 59(2):257–272.
- [Langdon et al., 2011] Langdon, W. B., Yoo, S., y Harman, M. (2011). Formal concept analysis on graphics hardware. En *CLA*, pp. 413–416. Citeseer.
- [Li y Zhang, 2018] Li, Y. y Zhang, Z. (2018). Parallel computing: review and perspective. En *2018 5th International Conference on Information Science and Control Engineering (ICISCE)*, pp. 365–369. IEEE.

- [Liu y Wang, 2013] Liu, B. y Wang, C. (2013). Association rule discovery based on formal concept analysis. En *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pp. 884–887. IEEE.
- [Liu et al., 2017] Liu, Z., Li, B., Pei, Z., y Qin, K. (2017). Formal concept analysis via multi-granulation attributes. En *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pp. 1–6. IEEE.
- [Luna et al., 2017] Luna, J. M., Padillo, F., Pechenizkiy, M., y Ventura, S. (2017). Apriori versions based on mapreduce for mining frequent patterns on big data. *IEEE transactions on cybernetics*, 48(10):2851–2865.
- [Ma et al., 2010] Ma, C., Zhang, Y., Wang, X., y Shen, H. (2010). A parallel algorithm based on dividing of closure system for structuring concept lattice. En *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, volumen 1, pp. 198–201. IEEE.
- [Min y Kim, 2019] Min, W. K. y Kim, Y. K. (2019). Soft concept lattice for formal concept analysis based on soft sets: Theoretical foundations and applications. *Soft Computing*, 23(19):9657–9668.
- [Naiouf y De Giusti, 2002] Naiouf, M. y De Giusti, A. E. (2002). Escalabilidad y balance de carga en sistemas paralelos. En *IV Workshop de Investigadores en Ciencias de la Computación*.
- [Qi et al., 2011] Qi, J.-J., Wei, L., y Hong, H.-Y. (2011). Concept lattice construction about many-valued contexts. En *2011 International Conference on Machine Learning and Cybernetics*, volumen 3, pp. 1124–1129. IEEE.
- [Sarmah et al., 2015] Sarmah, A. K., Hazarika, S. M., y Sinha, S. K. (2015). Formal concept analysis: current trends and directions. *Artificial Intelligence Review*, 44(1):47–86.
- [Vojtschovsky, 2003] Vojtschovsky, P. (2003). Investigation of subalgebra lattices by means of hasse constants. *algebra universalis*, 50(1):7–26.
- [Xiao y Hu, 2019] Xiao, W. y Hu, J. (2019). Mrclose: A parallel algorithm for closed frequent itemset mining based on mapreduce. En *Proceedings of the 2019 International Conference on Robotics Systems and Vehicle Technology*, pp. 7–13.