

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“USO DE HISTORIAL DE LBD’S PARA SELECCIONAR
POLÍTICA DE REINICIO EN SOLVERS DE OPTIMIZACIÓN
PSEUDO BOOLEANA BASADOS EN CDCL”

AMANDA PAZ SALINAS PINTO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN INFORMÁTICA

Profesor Guía: Roberto Asín
Profesor Correferente: Ricardo Ñanculef

Marzo - 2024

DEDICATORIA

*Para mi familia, amigos
y todas las personas que me acompañaron estos años*

AGRADECIMIENTOS

En primer lugar, me gustaría expresar mi profundo agradecimiento a mi familia, en especial a mi madre, Claudia Pinto, y a mi padre, Iván Salinas, por brindarme la oportunidad de estar aquí, así como por su inquebrantable apoyo y orientación a lo largo de todos mis años de vida. También quiero reconocer a mis hermanas, quienes han compartido conmigo tanto alegrías como enojos, y agradecerles por su cariño incondicional. Un especial reconocimiento a Francisco Olivares por su cariño, por las papas fritas y las risas que nunca faltan.

Quiero agradecer a mi pareja, Felipe Labra, que fue un pilar fundamental durante toda mi vida universitaria, agradezco su constante apoyo y compañía, por que es bonito crecer y aprender juntos cada día.

Quiero agradecer a mis amigas del colegio, Valentina Espinoza, mi compañera de puesto y una de las razones por que la elegí esta universidad. A Sofía Tapia, por que a pesar de haber tomado caminos diferentes, la llevo siempre presente. A mis amigos de ciencias, a quienes conocí en la universidad, a la gente de la sala de mate. Cada uno de ustedes ha aportado a mi vida universitaria con experiencias inolvidables, risas y un invaluable apoyo en los días difíciles.

A mis amigos de *running* y cafecitos, Javier y Valentina, les agradezco por ser parte esencial en mi día a día. Por ayudarme a despejarme cuando lo necesitaba y hacerme trabajar cuando no quería, sin ustedes esto no seria posible.

Un agradecimiento especial a mi profesor guía, Roberto Asín, por su constante acompañamiento durante la elaboración de mi memoria, mi estancia en Berkeley y por sus valiosos consejos en mis crisis existenciales. Aprecio su paciencia, orientación y motivación en el mundo de la investigación. Gracias por confiar en mí.

Finalmente, mi agradecimiento se extiende a todas las personas que han sido parte de mi vida, a aquellos que seguiré conociendo y a los que ya no están, pero cuya influencia ha contribuido a formar la persona que soy hoy.

RESUMEN

Resumen—*Pseudo Boolean Optimization* es un problema con una variedad de aplicaciones en el mundo real, sin embargo con una gran complejidad computacional al ser un problema *NP-Hard*. Existen diversos intentos de solucionarlo, entre ellos, los *solvers* basados en *Conflict Driven Clause Learning*, como *RoundingSat*, el cual ha implementado diversas heurísticas para mejorar su rendimiento, incluyendo técnicas de reinicio que evitan estancamientos y fomentan la exploración del espacio de búsqueda.

A pesar de los avances en las heurísticas para CDCL, en la actualidad un *solver* no puede dominar todo el conjunto de instancias de un problema. Por lo tanto, se vuelve esencial emplear el mejor *solver* para cada instancia específica. En este contexto, se ha desarrollado un selector de algoritmos para elegir entre técnicas de reinicio. Se ha empleado un modelo de LSTM, entrenado con una nueva caracterización para instancias PBO, basado en el historial de LBDs. La cual mostró ser competitiva al compararla con una implementación propia basada en el estado del arte, exhibiendo mejoras en los tiempos de resolución.

Palabras Clave— Optimización Pseudo Booleana; RoundingSat; Reinicio; Literal Block Distance

ABSTRACT

Abstract— Pseudo Boolean Optimization is a problem with a variety of real-world applications; however, it comes with significant computational complexity as it falls under the NP-Hard problem category. There have been various attempts to address it, including solvers based on Conflict Driven Clause Learning, such as RoundingSat, which has implemented various heuristics to enhance its performance. These heuristics include reset techniques that prevent stagnation and foster exploration of the search space.

Despite notable advancements in CDCL heuristics, currently, a solver cannot excel across the entire set of instances for a problem. Hence, it becomes essential to employ the best solver for each specific instance. In this context, an algorithm selector has been developed to choose between restart techniques. An LSTM model has been employed, trained with a new characterization for PBO instances based on the history of LBDs. This approach has proven to be competitive when compared to a state-of-the-art implementation, showcasing improvements in resolution times.

Keywords— Pseudo Boolean Optimization; RoundingSat; Restart; Literal Block Distance

GLOSARIO

CDCL: Conflic driven clause learning

CNF: Conjunctive normal form

LBD: Literal Block Distance

LSTM: Long short-term memory

ML: Machine learning

PBO: Pseudo boolean optimization.

RNN: Recurrent neural network

SBD: Single best solver

VBS: Virtual best solver

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Hipótesis	4
1.2 Objetivos	4
1.2.1 Objetivo general	4
1.2.2 Objetivos específicos	4
CAPÍTULO 2: MARCO TEÓRICO	5
2.1 Problemas NP-Hard	5
2.2 Optimización Pseudo Booleana	5
2.3 Conflict Driven Clause Learning	6
2.3.1 Heurísticas de Reinicio	8
2.3.2 Luby Restart	8
2.3.3 Glucose Restart	9
2.4 RoundingSAT Solver	10
2.5 Selección de algoritmos	12
2.5.1 Métricas de rendimiento	13
2.6 Configuración de algoritmos	14
2.7 Machine Learning	15
2.7.1 RNN (Recurrent neural network)	16
2.7.2 LSTM (Long Short-Term Memory)	16
2.7.3 Ensemble Learning	16
2.7.4 Árbol de decisión	17
2.7.5 Random Forest Classifier	17
2.7.6 Normalización Min-Max	17
CAPÍTULO 3: ESTADO DEL ARTE	19
3.1 Selección de Algoritmos en <i>solvers</i> basados en CDCL	19
3.1.1 SATzilla	19
3.1.2 SATfeatPy	20
3.1.3 LMPick	21
3.2 Machine Learning Restarts	23

CAPÍTULO 4: PROPUESTA DE SOLUCIÓN	25
4.1 Tecnologías	25
4.2 Instancias <i>PBO</i>	25
4.3 Construcción del portafolio	26
4.4 Construcción del <i>dataset</i>	33
4.4.1 Objetivo a predecir	35
4.5 Obtención de <i>features</i>	37
4.5.1 SATZilla	37
4.5.2 <i>LBD</i>	38
4.6 Selector de Algoritmos	39
CAPÍTULO 5: VALIDACIÓN DE LA SOLUCIÓN	41
5.1 Conjunto de validación	41
5.2 Procesamiento datos	43
5.3 Entrenamiento	45
5.4 Resultados	47
CAPÍTULO 6: CONCLUSIONES	51
6.1 Sobre los resultados	51
6.1.1 Construcción del <i>dataset</i>	51
6.1.2 Resultados del selector de algoritmos	51
6.2 Contribuciones	52
6.3 Sobre los objetivos	53
6.4 Limitaciones	54
6.5 Trabajo Futuro	54
ANEXOS	56
REFERENCIAS BIBLIOGRÁFICAS	57

ÍNDICE DE FIGURAS

1	Diagrama general del Algoritmo CDCL Fuente: Elaboración propia.	7
2	Algoritmo CDCL de RoundingSAT Fuente: [Devriendt <i>et al.</i> , 2021a]	11
3	Algoritmo de optimización de <i>RoundingSAT</i> Fuente: [Devriendt <i>et al.</i> , 2021a]	11
4	Descripción esquemática de la interacción entre las características de la instancia del problema, datos de rendimiento, construcción del selector y evaluación del rendimiento del selector. Fuente: [Kerschke <i>et al.</i> , 2019]	12
5	Conexiones entre selección de algoritmos y problemas relacionados. Fuente: [Kerschke <i>et al.</i> , 2019]	14
6	Características utilizadas para construir <i>SATzilla07</i> Fuente: [Xu <i>et al.</i> , 2008]	19
7	Características utilizadas para construir <i>LMPick</i> Fuente: [Haim y Walsh, 2009]	22
8	<i>Cactus-plot</i> de las políticas de reinicio <i>Luby</i> , <i>Glucose</i> Y <i>MLR</i> . Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: [Jia <i>et al.</i> , 2018]	24
9	Esquema del flujo de trabajo Fuente: Elaboración propia	25
10	<i>Cactus-plot</i> de las configuraciones de parámetros para la técnica de reinicio <i>Luby</i> . Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia	30
11	<i>Cactus-plot</i> de las configuraciones de parámetros para la técnica de reinicio <i>Glucose</i> . Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia	31
12	Matriz de complementariedad entre cada par de configuraciones de <i>Luby</i> . Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia	32
13	Matriz de complementariedad entre cada par de configuraciones de <i>Glucose</i> . Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia	33
14	<i>Cactus-plot</i> de los algoritmos del portafolio. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia	34
15	Matriz de complementariedad entre cada par de algoritmos del portafolio. Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia	35
16	Histograma de las veces que un algoritmo gana sobre una instancia Fuente: Elaboración propia	36

17	Histograma que refleja la frecuencia en que un algoritmo gana en una instancia, excluyendo aquellas en las que todos los algoritmos la resuelven en menos de 5 segundos. Estas ultimas se ven agrupadas en la categoría ``None"Fuente: Elaboración propia	36
18	Histograma de la cantidad de instancias por familia, los colores representan el algoritmo que gana esa instancia Fuente: Elaboración propia	37
19	Gráfico de las curvas de valores de LBD originales y normalizados a lo largo de la búsqueda Fuente: Elaboración propia	39
20	Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de entrenamiento Fuente: Elaboración propia	42
21	Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación Fuente: Elaboración propia	42
22	Matriz de correlación de las características de <i>SATZilla</i> Fuente: Elaboración propia	43
23	Gráfico de Información mutua entre las características de <i>SATZilla</i> y la variable objetivo Fuente: Elaboración propia	44
24	Gráfico de las curvas de valores de LBD originales para dos intancias distintas Fuente: Elaboración propia	45
25	Gráfico de las curvas de valores de LBD normalizados a lo largo de la búsqueda para dos instancias distintas Fuente: Elaboración propia	45
26	Gráfico del <i>accuracy</i> obtenido en cada iteración del entrenamiento Fuente: Elaboración propia	46
27	Gráfico de importancia de características del modelo <i>Random Forest</i> entrenado sobre características de <i>SATZilla</i> Fuente: Elaboración propia	47
28	Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación para el modelo entrenado con el historial de LBD Fuente: Elaboración propia	48
29	Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación para el modelo entrenado con las características de <i>SATZilla</i> Fuente: Elaboración propia	49
30	<i>Cactus-plot</i> de los selectores de algoritmos entrenados con LBD y <i>SATZilla</i> comparados con los VBS y SBS. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia	50

ÍNDICE DE TABLAS

1	Valores del <i>accuracy</i> y tiempo promedio computacional (en segundos) para el problema de satisfacibilidad sobre los conjuntos de características Fuente: [Provan-Bessell <i>et al.</i> , 2022]	21
2	Valores del <i>accuracy</i> y tiempo promedio computacional (en segundos) para el problema de categorización en familias de problemas sobre los conjuntos de características Fuente: [Provan-Bessell <i>et al.</i> , 2022]	21

3	Configuración de parámetros de los experimentos realizados con la técnica de reinicio <i>Luby</i> Fuente: Elaboración Propia.	27
4	Configuración de parámetros en los experimentos realizados con la técnica de reinicio <i>Glucose</i> Fuente: Elaboración Propia.	28
5	Cantidad de instancias resueltas por cada configuración de parámetros de <i>Glucose</i> Fuente: Elaboración Propia.	29
6	Cantidad de instancias resueltas por cada configuración de parámetros de <i>Luby</i> Fuente: Elaboración Propia.	29
7	Cantidad de instancias resueltas por cada algoritmo del portafolio, incluye el <i>Virtual Best Solver</i> Fuente: Elaboración Propia.	34
8	Descripción de capas de la red neuronal creada con la cantidad de parámetros a entrenar por cada una Fuente: Elaboración propia	39
9	Métricas <i>accuracy</i> , <i>F1-Score</i> y $\hat{m}$ para los distintos selectores de algoritmos entrenados para la caracterización de <i>SATzilla</i> y LBD Fuente: Elaboración propia	47
10	Cantidad de instancias resueltas por los selectores de algoritmos entrenados para las caracterizaciones de <i>SATzilla</i> y LBD, comparados con su VBS y SBS sobre el conjunto de validacion Fuente: Elaboración propia	49

INTRODUCCIÓN

La teoría de la complejidad computacional clasifica problemas en categorías como P, NP, NP-Completo y NP-Hard [Arora y Barak, 2009]. Estos últimos son considerados problemas difíciles de resolver, ya que el tiempo necesario para su resolución crece rápidamente con el tamaño de las instancias [Woeginger, 2003].

Un problema con esta clasificación es *Pseudo Boolean Optimization* (PBO), el cual es un problema de optimización donde las funciones objetivo contienen términos pseudo-booleanos [Boros y Hammer, 2002]. Resolver este problema implica encontrar la asignación óptima de valores a las variables binarias que minimice o maximice la función objetivo. Dada su complejidad teórica, su resolución requiere algoritmos eficientes y técnicas de optimización avanzadas.

Pseudo Boolean Optimization ha ganado reconocimiento debido a que ofrece mayor flexibilidad a la hora de modelar situaciones de la vida real, razón por la cual tiene una gran cantidad de aplicaciones que pueden ser representadas como instancias de este problema. Actualmente, las técnicas asociadas a PBO son cruciales en diversos campos, por ejemplo, la minería de datos, planificación, transporte, gestión e ingeniería [Liu et al., 2020].

En la actualidad el problema se beneficia de los avances e investigaciones en técnicas de programación lineal entera y en el problema de satisfaciibilidad booleana (SAT). Se emplean algoritmos de búsqueda basados en *Conflict Driven Clause Learning* (CDCL), junto con estrategias heurísticas diseñadas por expertos del dominio. Estas estrategias incluyen decisiones para guiar la búsqueda, limpieza de cláusulas y políticas de reinicio [Liu et al., 2020]. Entre los solvers PBO, se encuentra *RoundingSAT*, el cual, basado en el algoritmo CDCL para la resolución de instancias PBO, se distingue por la implementación de heurísticas específicas diseñadas para potenciar su eficacia [Elffers y Nordström, 2018].

Las políticas de reinicio son heurísticas que tienen impacto en la eficiencia de los solvers basados en CDCL, ya que promueven la exploración del espacio de búsqueda y evita que estos queden estancados [Pipatsrisawat y Darwiche, 2009]. Aunque se han propuestos diversos esquemas de reinicio, la literatura actual indica que ningún solver es capaz de dominar sobre todas las instancias de un problema, por lo que se necesita adaptar el solver a utilizar a la instancia que se está intentando resolver.

Para esto, se propone construir un sistema seleccionador que elija la técnica de reinicio según las instancias de problemas PBO. Para lograrlo, se investigarán caracterizaciones de instancias PBO y se entrenarán modelos utilizando algoritmos de *machine learning*, como las redes neuronales *Long short-term memory* (LSTM).

Estructura del Documento

La principal contribución de esta investigación es el uso de una nueva caracterización para

instancias PBO, a partir de series de los valores de LBDs obtenidos durante la ejecución del solver *RoundingSAT*. Además, la construcción de un sistema selector para predecir la técnica de reinicio apropiada a utilizar por *RoundingSAT* en instancias PBO.

Esto se ve reflejado en la siguiente estructura

- En el *Capítulo 1: Definición del problema* se establece y delimita la problemática. Se formula la hipótesis a trabajar y los objetivos generales y específicos de la investigación.
- En el *Capítulo 2: Marco teórico*, se definen los conceptos fundamentales relacionados al problema.
- En el *Capítulo 3: Estado del arte* se detalla la literatura actual relacionada al problema, sintetizando los estudios mas relevantes.
- En el *Capítulo 4: Propuesta de solución* Se describe detalladamente la metodología utilizada para la investigación.
- En el *Capítulo 5: Validación de la solución* Se presentan resultados y análisis derivados de la validación, y su comparación con el estado del arte.
- En el *Capítulo 6: Conclusiones* Se presentan las conclusiones, destacando limitaciones, contribuciones y posible trabajo futuro.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

Resolver el problema de optimización de pseudo-booleanos (PBO) es de suma importancia debido a su capacidad de representar de manera general problemas complejos con aplicaciones en diversos campos de la vida real [Liu *et al.*, 2020]. En la actualidad, la resolución de PBO se beneficia de algoritmos basados en *Conflict Driven Clause Learning* (CDCL), que exploran el espacio de búsqueda mediante búsquedas en árbol. Diversos *solvers*, como RoundingSat, se centran en abordar este problema implementando algoritmos CDCL y otras heurísticas [Elffers y Nordström, 2018].

En la práctica, los algoritmos basados en CDCL se pueden quedar atascados en un mínimo local [Devriendt *et al.*, 2021a]. Una solución a este problema es utilizar técnicas de reinicio para fomentar la exploración del espacio de búsqueda, lo que implica limpiar el árbol de búsqueda actual y comenzar de nuevo sin olvidar las cláusulas y variables ya aprendidas. Aunque pueda parecer contraproducente, los *solvers* que hacen reinicios frecuentes son significativamente más rápidos que aquellos que optan por no realizarlos [Liang *et al.*, 2018].

En este contexto, se han propuesto diversos esquemas de reinicio, ya sea estáticos como los utilizados por RoundingSAT [Huang, 2007] o dinámicos, como en Glucose [Audemard y Simon, 2009]. Sin embargo, la literatura actual indica que un *solver* no es capaz de dominar sobre todo el conjunto de instancias de un problema, lo que promueve la especialización a grupos de instancias determinados y la creación de sistemas selectores capaces de indicar el *solver* apropiado para cada instancia [Kerschke *et al.*, 2019].

La construcción de un sistema selector requiere de una caracterización adecuada de las instancias del problema para entrenar el modelo a través de algoritmos de *machine learning* [Kerschke *et al.*, 2019]. Para esto se suelen utilizar datos de fácil obtención, como la cantidad de variables y restricciones del problema, entre otros. En la comunidad de Programación con Restricciones (CP), es particularmente influyente la propuesta de SATZilla [Xu *et al.*, 2008], sin embargo los resultados obtenidos se pueden atribuir a su capacidad de clasificar instancias en tipos de problemas, en lugar de extraer características propias de cada instancia [Provan-Bessell *et al.*, 2022].

Una opción no estudiada para la caracterización de instancias es utilizar los valores de un indicador dinámico de búsqueda llamado *Literals Blocks Distance* (LBD), obtenidos durante la ejecución del algoritmo CDCL. Estos valores se utilizan como heurística para determinar la calidad de las restricciones aprendidas [Audemard y Simon, 2009]. Se han utilizado como guía para políticas de reinicio en Glucose [Audemard y Simon, 2012] y también se han empleado para entrenar modelos de *machine learning* que predicen la calidad futura de la búsqueda [Jia *et al.*, 2018].

En este trabajo, para mejorar el rendimiento de los algoritmos CDCL, se propondrán algorit-

mos de *machine learning* para determinar la política de reinicio adecuada, dada una instancia de PBO. El foco se encuentra en construir sistemas selectores que se adapten a las instancias de los problemas y caracterizar las instancias de forma tal que el entrenamiento de los modelos selectores sea significativo.

1.1. Hipótesis

Long Short-Term Memory Neural Networks (LSTM), entrenadas sobre el historial de LBDs de *RoundingSat*, son capaces de seleccionar la técnica de reinicio apropiada para cada instancia del problema *Pseudo-Boolean Optimization*.

1.2. Objetivos

1.2.1. Objetivo general

El **objetivo general** consiste en el diseño e implementación de un selector de algoritmos para predecir técnica de reinicio del *solver RoundingSat* para instancias del problema *Pseudo Boolean Optimization* (PBO)

1.2.2. Objetivos específicos

- Implementar técnicas de reinicio en el *solver RoundingSat*.
- Analizar el rendimiento de técnicas de reinicio en la búsqueda del *solver RoundingSat*.
- Proponer una caracterización para instancias PBO.
- Implementar un selector de algoritmos para escoger la técnica de reinicio del *solver RoundingSAT*, dada una instancia de PBO.
- Evaluar el desempeño y comparar resultados con el estado del arte.

CAPÍTULO 2

MARCO TEÓRICO

2.1. Problemas NP-Hard

La teoría de la complejidad computacional proporciona herramientas y técnicas para clasificar y caracterizar problemas en función de sus características. Entre las clases de complejidad propuestas se encuentran los conjuntos de problemas P, NP, NP-Hard, P-SPACE, EXP, etc. La pertenencia de un problema a estos conjuntos nos permite comprender ciertas características computacionales de los problemas, determinar enfoques de solución y aterrizar expectativas sobre, por ejemplo, el tamaño de instancias que pueden ser tratadas en un tiempo aceptable, garantías de terminación y completitud, entre otras.

Los problemas NP-hard son aquellos problemas que generalizan la complejidad de los problemas NP. Esto es, existe una reducción de tamaño polinomial de cualquier problema NP a otro problema NP-hard. Un problema pertenece a NP si existe un certificado de tamaño polinomial para la resolución del problema. Informalmente, esto significa que se puede verificar en tiempo polinomial si una solución para el problema es válida o no. Un problema es NP-completo si pertenece tanto a P, como a NP-hard. En teoría de la computación, se conjetura que los problemas NP-Completo son intratables, es decir, se cree que no existe un algoritmo polinomial que los resuelva [Arora y Barak, 2009], a menos que $P=NP$.

La importancia de los problemas NP-Hard se debe a que estos pueden ser usados como lenguajes de modelado para una amplia gama de desafíos computacionales en áreas como la optimización combinatoria [Nemhauser y Wolsey, 1988], la teoría de grafos [Diestel, 2017], entre otros. Algunos ejemplos comunes incluyen el problema del viajante (*Traveling Salesman Problem*) [Applegate et al., 2007], el problema de la mochila (*Knapsack Problem*) [Martello y Toth, 1991] y el problema de satisfacción booleana (*Boolean Satisfiability Problem*) [Hoos, 2005].

2.2. Optimización Pseudo Booleana

El problema de Optimización Pseudo-Booleana (PBO, por sus siglas en inglés) se centra en la optimización de una función pseudo booleana, la cual se define como un mapeo de una familia de subconjuntos binarios finitos a los reales, como se muestra en la ecuación 1. Donde f se puede representar como una combinación lineal o polinomial de variables booleanas [Nordström, 2021].

$$f : \{0, 1\}^n \rightarrow \mathbb{R} \quad (1)$$

Este tipo de representación permite realizar una conexión natural entre estas funciones y la optimización no lineal binaria [Boros y Hammer, 2002], lo que brinda mayor flexibilidad para modelar problemas complejos.

Resolver este problema implica encontrar la asignación óptima de valores a variables binarias para minimizar o maximizar la función objetivo dada, cumpliendo un conjunto de restricciones, las cuales en su forma lineal entera se expresan de la siguiente manera.

$$\sum_i a_i l_i \bowtie A \quad (2)$$

Donde:

- $\bowtie$ representa las operaciones: $\geq, \leq, >, =, <$
- $a_i, A \in \mathbb{Z}$
- Los literales l_i representan a las variables x_i o $\bar{x}_i$

El problema toma un rol importante en la optimización de modelos de diversas áreas, por ejemplo, minería de datos, planificación, transporte, gestión e ingeniería [Liu et al., 2020]. Sin embargo, debido a que se encuentra dentro de la categoría NP-Hard [Boros y Hammer, 2002], es difícil encontrar algoritmos lo suficientemente efectivos en instancias grandes.

En la actualidad, se dispone de *solvers* que emplean técnicas específicas para aprovechar las propiedades de estos problemas, utilizando métodos de búsqueda en árbol para la exploración del espacio de búsqueda. Principalmente, éstos incluyen técnicas basadas en *Conflict Driven Clause Learning* (CDCL) [Zhang et al., 2001] y *Branch & Bound* [Lawler y Wood, 1966].

2.3. Conflict Driven Clause Learning

El aprendizaje de cláusulas por conflictos, también conocido como CDCL (*Conflict-Driven Clause Learning*), es una técnica fundamental utilizada en la resolución moderna de problemas de satisfacción booleana, como SAT. Este método se basa en la exploración del espacio de búsqueda a través del análisis de conflictos que permiten aprender nuevas cláusulas durante la ejecución del programa [Nordström, 2021].

El algoritmo de resolución CDCL se lleva a cabo en etapas, tal como se muestra en la figura 1. Comienza con una asignación inicial de valores a una variable. A continuación, sigue un proceso de propagación unitaria que revisa las restricciones del problema para forzar asignaciones necesarias que eviten la falsificación de cláusulas. Una vez que este proceso concluye,

se verifica si se ha producido un conflicto (i.e. alguna restricción es falsa bajo la asignación actual de las variables). En caso afirmativo, se realiza un análisis del conflicto para aprender una nueva restricción y evitar en el futuro la asignación que causó el conflicto. Finalmente, se retrocede a un punto anterior al conflicto (*backjump*) y se vuelve a realizar la propagación. Por otro lado, si no hay conflicto, se revisa si es el momento de realizar un reinicio. De ser así, se eliminan las asignaciones realizadas hasta el momento y se reinicia la búsqueda. En caso contrario, se toma una nueva decisión y se itera.

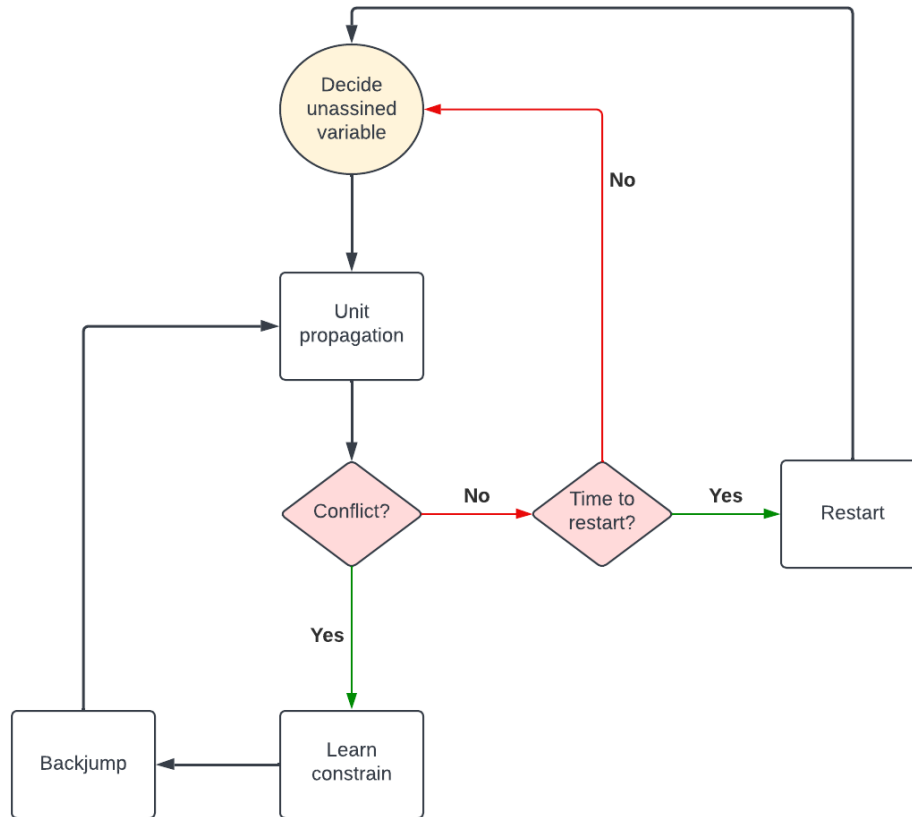


Figura 1: Diagrama general del Algoritmo CDCL Fuente: Elaboración propia.

Si bien, este algoritmo se propuso originalmente para SAT, también se ha extendido con éxito al problema de optimización pseudo-booleana. Para ello, existen los siguientes enfoques:

- **Conversión a cláusulas:** Esto significa transformar las restricciones de PBO a su forma CNF. Utilizado por *MiniSAT+* [Eén y Sörensson, 2006] y *Naps* [Sakai y Nabeshima, 2015]
- **Razonamiento nativo:** Esto significa adaptar el aprendizaje de cláusulas a las restricciones pseudo-booleanas. Utilizado por *Roundingsat* [Elffers y Nordström, 2018]

Para cada componente de este algoritmo se han propuesto diversos y sofisticados algoritmos que han impactado fuertemente en el rendimiento de estos *solvers*. En la actualidad, estos *solvers* son capaces de decidir fórmulas de lógica proposicional Booleana con millones de variables y decenas de millones de cláusulas. Así, por ejemplo, existen heurísticas especializadas para decidir sobre qué literal se realizará una decisión y con qué polaridad [Biere y Fröhlich, 2015], propagar eficientemente [Bordeaux *et al.*, 2006], analizar efectivamente un conflicto y aprender un nuevo lema simplificado [Zhang *et al.*, 2001], políticas de limpiado de la base de datos de cláusulas [Audemard *et al.*, 2011] y políticas de reinicio [Audemard y Simon, 2009] [Huang, 2007]. Considerando el foco de este trabajo, a continuación se elabora sobre este último aspecto, las heurísticas de reinicio.

2.3.1. Heurísticas de Reinicio

El algoritmo CDCL, descrito anteriormente, se caracteriza por su habilidad para adaptarse a situaciones conflictivas y aprender de ellas. En este contexto, una faceta esencial de su dinámica se encuentra en la implementación del reinicio, un proceso estratégico que potencia su eficacia en la resolución de problemas complejos. Esta heurística tiene como objetivo reiniciar el proceso de búsqueda en momentos estratégicos para evitar estancamientos y mejorar la capacidad del algoritmo para encontrar soluciones de manera más efectiva [Pipatsrisawat y Darwiche, 2009]. Estas técnicas se clasifican en:

- **Estáticas:** aquellas que reinician luego de una cantidad definida de conflictos, sin considerar la calidad de la búsqueda. Entre ellas está el *solver RoundingSAT* con la estrategia *luby* [Devriendt *et al.*, 2021a]
- **Dinámicas:** aquellas que toman en cuenta la calidad de las restricciones aprendidas hasta el momento. Entre ellas se encuentra el *solver Glucose* para SAT [Audemard y Simon, 2009]. Sin embargo no han sido muy estudiadas en PBO.

Entre los esquemas propuestos, se detallan a continuación los reinicios basados en *luby* y *glucose*.

2.3.2. Luby Restart

Esta estrategia se fundamenta en la secuencia geométrica de *Luby*, la cual se utiliza para establecer la frecuencia de los reinicios. Se define como la secuencia $t_1, t_2, t_3, \dots$. Donde el término t_i , con $i \in \mathbb{N}$, se expresa de la siguiente manera [Huang, 2007] .

$$t_i = t_{i-1}^g \cdot g^{i-1} \quad (3)$$

Con $g \in \mathbb{Z}^+$.

Luego, la estrategia define un contador que indica la cantidad de conflictos que deben ocurrir para desencadenar un reinicio. Este contador se obtiene a partir de la multiplicación entre un valor constante $u \in \mathbb{Z}^+$ y el término de la serie *luby* t_i . Es decir, el largo del reinicio i , denotado por r_i se define de la siguiente manera.

$$r_i = u \cdot (t_i)_{i \in \mathbb{N}} \quad (4)$$

Los parámetros a configurar son los siguientes.

1. **Luby Base (u):** $u \in \mathbb{Z}^+$, corresponde a el valor constante por el cual se multiplican los valores de la serie.
2. **Luby multiplier (g):** $g \in \mathbb{Z}^+$, corresponde al valor de crecimiento de la secuencia t_i .

Se ha utilizado con éxito con distintos valores de su parámetro *luby base*. Utilizado en el portafolio de *LMPick* con valores de 32 y 512 [Haim y Walsh, 2009]. Por otro lado, *RoundingSat* lo define por defecto con un valor de 100 [Devriendt et al., 2021a].

No se encuentran resultados de variaciones a su parámetro *luby multiplier*, sin embargo, se define por defecto con un valor de 2.

2.3.3. Glucose Restart

En esta técnica, la decisión de realizar o no un reinicio se basa en la calidad de las restricciones aprendidas. Cuando la calidad disminuye, indica que el *solver* está explorando el espacio incorrecto [Berre y Wallon, 2021]. La calidad de las restricciones se evalúa mediante su *Literals Blocks Distance* (LBD), cuya definición se presenta a continuación [Audemard y Simon, 2009].

Definición 1 (Literals Blocks Distance (LBD)) Dada una clausula C , y una partición de sus literales en n subconjuntos de acuerdo a la asignación actual, tal que los literales están particionados con respecto a su nivel de decisión (i.e. el número de decisiones tomadas antes de propagar el literal). El LBD de C es exactamente n .

El objetivo de esta técnica es minimizar el número de decisiones antes de alcanzar un conflicto. Para lograr esto, se lleva a cabo un reinicio si los LBDs de las restricciones aprendidas más recientemente son altas en comparación con el promedio histórico de los LBDs [Audemard y Simon, 2012]. Los parámetros a configurar son los siguientes.

1. **Glucose window (w)** $w \in \mathbb{Z}$, corresponde a la cantidad de restricciones recientes que se tomaran en cuenta.
2. **Glucose multiplier (m)**: $m \in \mathbb{R}$, $0 \leq m \leq 1$. corresponde al porcentaje con el cual se comparará el promedio de las restricciones recientes con el histórico

Los solvers de *Glucose* para SAT, han mostrado exitosos resultados, ganando la competencia de SAT del 2011. Se realizaron estudios de los parámetros óptimos sobre *Glucose 2.0*, obteniendo comportamiento distintos sobre los conjuntos satisfacibles (SAT) e insatisfacibles (UNSAT) [Audemard y Simon, 2012]. Los mejores resultados se obtuvieron para instancias SAT, con un *glucose window* de 100 y un *glucose multiplier* de 0,7. Por otro lado, para instancias UNSAT un *glucose window* de 50 y un *glucose multiplier* de 0,8, mostraron mejoras.

2.4. RoundingSAT Solver

Uno de los solver actuales para la resolución del problema de optimización pseudo-booleana es *RoundingSAT*, el cual realiza una implementación de razonamiento nativo del algoritmo CDCL para PBO [Elffers y Nordström, 2018]. Además, utiliza otras heurísticas como la integración con un software de programación lineal entera [Devriendt et al., 2021a], técnicas mas avanzadas de propagación unitaria [Devriendt, 2020] y técnicas de optimización basadas en núcleos de insatisfacibilidad [Devriendt et al., 2021b]. Estas técnicas permiten acelerar la búsqueda de soluciones y mejora su capacidad de manejar problemas mas grandes y complejos.

Una esquema general del algoritmo CDCL que implementa *RoundingSat* se muestra en la figura 2. Inicia con la toma de decisión sobre una variable y continúa con la propagación unitaria que asigna valores a las variables según las restricciones del problema. En caso de conflicto, se analiza la situación y se aprenden nuevas restricciones para mejorar la búsqueda a futuro. Una vez se aprende una nueva restricción, se realiza un *backjump*, es decir, se devuelve a través del árbol de decisiones hasta llegar a un punto sin conflicto. Posteriormente, se reanuda la propagación desde la última asignación de variable.

Después de un número predefinido de conflictos, se evalúa si es hora de un reinicio o bien si hay que limpiar el conjunto de restricciones aprendidas.

Algorithm 1 Simplified version of conflict-driven PB search with linear programming integration (LP-specific code highlighted in grey).

Data: pseudo-Boolean formula φ ; constraint database ψ ; linear relaxation $\widehat{\psi}$
Result: solution ρ to φ or empty assignment $\emptyset$ if φ is unsatisfiable

- 1 initialize ρ to the empty assignment $\emptyset$
- 2 initialize last propagated value to 0 for all variables
- 3 **while** true **do**
- 4 **while** some constraint $C \in \psi$ has an unassigned literal ℓ_i with coefficient $c_i > \text{slack}(C, \rho)$ **do**
- 5 $\rho \leftarrow \rho \cup \{\ell_i\}$
- 6 record C as the reason for ℓ_i
- 7 update last propagated value for variable corresponding to ℓ_i
- 8 **if** some $C \in \psi$ has $\text{slack}(C, \rho) < 0$ **then**
- 9 perform conflict analysis on C and the current reason constraints to derive a new constraint C_{learned}
- 10 **if** C_{learned} implies $0 \geq 1$ **then return** $\emptyset$
- 11 **else**
- 12 add C_{learned} to ψ
- 13 remove assignments from ρ in reverse order until $\text{slack}(C_{\text{learned}}, \rho) \geq 0$
- 14 **else if** $\widehat{\psi}$ is rationally infeasible under ρ **then**
- 15 extract Farkas multipliers to construct C_{Farkas}
- 16 add C_{Farkas} to ψ
- 17 **else if** ρ assigns all variables **then return** ρ
- 18 **else if** time to restart **then** $\rho \leftarrow \emptyset$
- 19 **else if** time to clean ψ **then** remove some learned constraints from $\psi \setminus \varphi$
- 20 **else** decide to assign some unassigned variable to its last propagated value

Figura 2: Algoritmo CDCL de RoundingSAT Fuente: [Devriendt et al., 2021a]

Algorithm 2 Simplified version of pseudo-Boolean optimization with linear programming integration (LP-specific code highlighted in grey).

Data: pseudo-Boolean formula φ ; objective function $f: \sum_i c_i \ell_i$
Result: solution ρ_{best} to φ minimizing f or empty assignment $\emptyset$ if φ is unsatisfiable

- 1 initialize ψ to φ
- 2 initialize $\widehat{\psi}$ to φ
- 3 initialize the LP solver with $\widehat{\psi}$ and f
- 4 call Algorithm 1 with $(\varphi, \psi, \widehat{\psi})$ and let ρ be the assignment returned
- 5 $\rho_{\text{best}} \leftarrow \rho$
- 6 **while** $\rho \neq \emptyset$ **do**
- 7 $\rho_{\text{best}} \leftarrow \rho$
- 8 add upper bound constraint $\sum_i c_i \ell_i \leq f(\rho_{\text{best}}) - 1$ to ψ and $\widehat{\psi}$
- 9 call Algorithm 1 with $(\varphi, \psi, \widehat{\psi})$ and let ρ be the assignment returned
- 10 **return** ρ_{best}

Figura 3: Algoritmo de optimización de RoundingSAT Fuente: [Devriendt et al., 2021a]

El algoritmo de la figura 3 muestra el proceso de optimización de la solución. Este genera una solución inicial, asignando valores a las variables del problema. Luego, se llama al algoritmo CDCL para determinar la factibilidad de la asignación proporcionada. En caso de que sea factible, se entrega una restricción que permite acotar el valor de la función objetivo. Este proceso se repite de forma iterativa, hasta encontrar un mínimo o máximo.

2.5. Selección de algoritmos

Se ha observado que, en problemas computacionales donde existen diversos algoritmos de alto rendimiento, no hay ninguno que domine sobre todo el conjunto de instancias de problemas existentes. Al contrario, cada algoritmo rinde mejor sobre conjuntos específicos de instancias. Al notar esta complementariedad, surge el problema de selección de algoritmos, el cual, determina cuál de los algoritmos se espera que obtenga un mejor rendimiento sobre la instancia de un problema. Formalmente, dado un conjunto I de instancias de un problema P , un conjunto $A = \{A_1, \dots, A_n\}$ de algoritmos para P y una métrica $m : A \times I \rightarrow \mathbb{R}$ que mide el rendimiento de cualquier algoritmo $A_j \in A$ sobre el conjunto de instancias I , se construye un selector S que mapea una instancia $i \in I$ a un algoritmo $S(i) \in A$, de manera que el rendimiento de S en I sea óptimo de acuerdo a la métrica definida [Kerschke *et al.*, 2019].

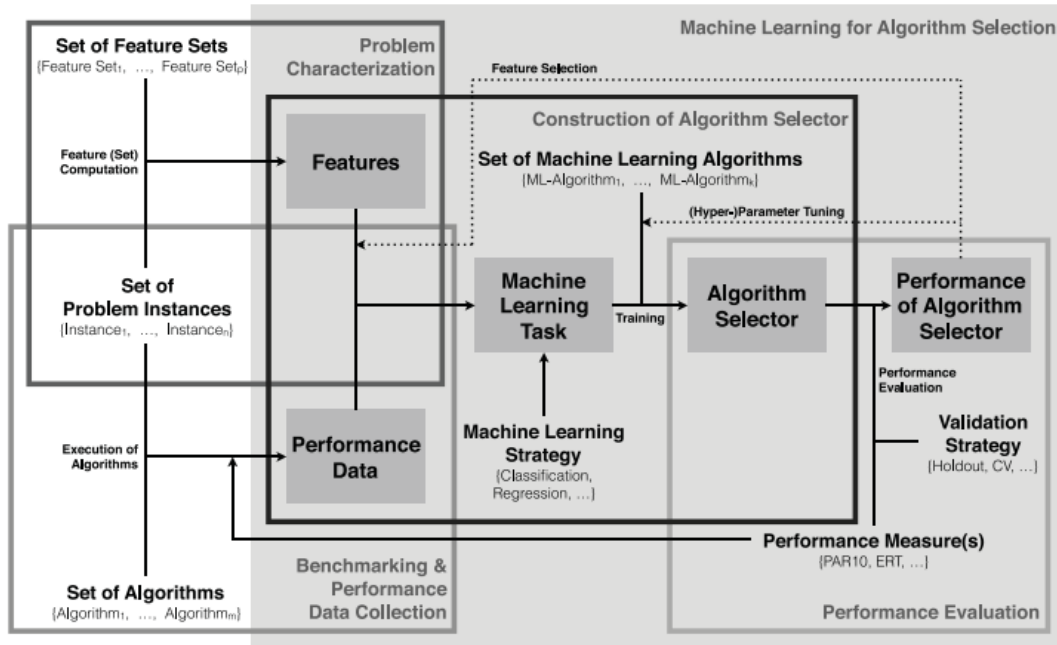


Figura 4: Descripción esquemática de la interacción entre las características de la instancia del problema, datos de rendimiento, construcción del selector y evaluación del rendimiento del selector. Fuente: [Kerschke *et al.*, 2019]

La figura 4 proporciona una representación visual de la relación entre los componentes que participan en la creación de un selector de algoritmos, las principales tareas son:

Caracterización del problema: Esto se logra a través de la obtención de características que presentan información relevante de las instancias. Para ello, se realiza el cálculo de características sobre un conjunto de instancias del problema.

Benchmarking: Consiste en ejecutar cada uno de los algoritmos del conjunto sobre el conjunto de instancias para recopilar su rendimiento según una métrica, como el tiempo de resolución, calidad de la solución, etc.

Construcción de un selector de algoritmos: Implica seleccionar un algoritmo del conjunto de algoritmos de *machine learning*. Éste se construye mediante la caracterización de las instancias y los datos de rendimiento obtenidos en el paso anterior. El selector asigna a cada instancia un algoritmo que optimiza el rendimiento.

Evaluación del rendimiento: Se utilizan métricas para evaluar el rendimiento del algoritmo selector construido.

2.5.1. Métricas de rendimiento

Para poder evaluar el rendimiento de los sistemas de selección de algoritmos, se compara su rendimiento con dos líneas base de referencia fundamentales:

Single Best Solver (SBS): Representa el rendimiento del algoritmo individual, en el portafolio, que, en promedio, rinde mejor sobre todas las instancias de entrenamiento. Este enfoque demuestra el máximo rendimiento posible sin la intervención de la selección de algoritmos.

Virtual Best Solver (VBS): Refleja el rendimiento de un selector que toma decisiones perfectas y siempre selecciona el mejor algoritmo para cada instancia, sin incurrir en sobrecarga adicional.

Se define una métrica de rendimiento para un selector de algoritmos, a partir de la métrica m para cada par $(algoritmo, instancia)$.

$$m_s = \sum_{i \in I} m(s(i), i) \quad (5)$$

Una forma común de medir rendimiento de tiempo de ejecución es a través del tiempo de ejecución promedio penalizado (PAR10): el promedio de tiempo de ejecución sobre todas

las instancias, donde los algoritmos se ejecutan con un tiempo límite y se penaliza con 10 veces el tiempo límite, si no pueden resolver la instancia.

El rendimiento de los sistemas de algoritmos varían para distintos escenarios, Por lo que se normaliza el rendimiento m_s de un selector de algoritmos, por el rendimiento del SBS y el VBS, como un costo a minimizar y se mide la brecha entre ambos. Se muestra a continuación

$$\hat{m}_s = \frac{m_s - m_{VBS}}{m_{SBS} - m_{VBS}} \quad (6)$$

Donde 0 corresponde a un rendimiento perfecto, equivalente al VBS y 1 corresponde al rendimiento del SBS. Un selector de algoritmos útil obtendrá un valor de esta métrica que se encuentre entre esos valores. Algo mayor a 1 significa que utilizar el SBS sería una mejor estrategia y, por tanto, el selector no tiene utilidad.

Existen tareas relacionadas con la selección de algoritmos, como, por ejemplo, la configuración de algoritmos, el diseño de portafolios, entre otros. A continuación, se desarrollará el concepto de configuración de algoritmos.

2.6. Configuración de algoritmos

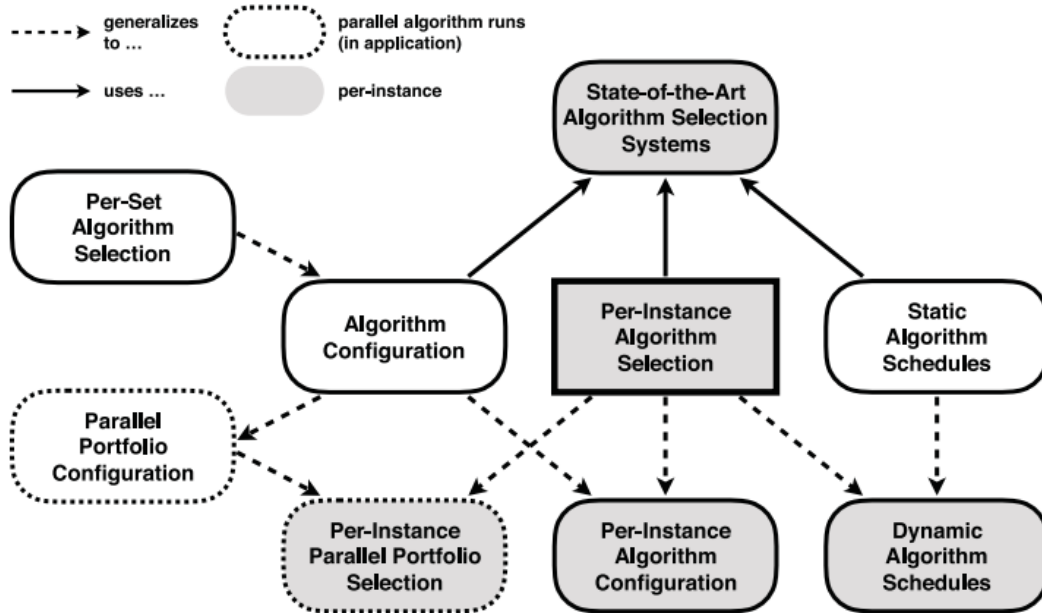


Figura 5: Conexiones entre selección de algoritmos y problemas relacionados. Fuente: [Kerschke et al., 2019]

Consiste en encontrar la configuración óptima de parámetros para un algoritmo dado. Formalmente, dado un algoritmo A , cuyo rendimiento se ve afectado por la configuración de sus parámetros $p = (p_1, \dots, p_k)$, un conjunto C de posibles valores para p , un conjunto de instancias del problema I y una métrica de rendimiento m , el objetivo es encontrar una configuración $c^* \in C$ para A , que logre un rendimiento óptimo en I según la métrica utilizada [Kerschke et al., 2019].

En la figura 5 se puede visualizar la relación entre los problemas relacionados con la selección de algoritmos, donde la configuración de algoritmos, por instancia, se puede ver como una generalización de la selección de algoritmos. En este caso, el conjunto de algoritmos que forman la base para la selección incluye todas las configuraciones (válidas) de un solo algoritmo parametrizado.

La diferencia clave con la selección de algoritmos radica en que este conjunto puede ser muy extenso, ya que surge de la explosión (posiblemente exponencial o infinita) de considerar todos los valores de los parámetros individuales, que en algunos casos pueden tomar valores continuos, dando lugar a conjuntos de configuraciones de algoritmos incontables sobre los cuales se debe optimizar [Kerschke et al., 2019].

2.7. Machine Learning

Machine Learning es un área en constante evolución, que trata con algoritmos computacionales diseñados para emular la inteligencia humana. En la actualidad, desempeña un papel fundamental en numerosas aplicaciones, desde el reconocimiento de patrones, la ingeniería espacial, finanzas, hasta el entretenimiento y aplicaciones médicas [El Naqa y Murphy, 2015].

En términos generales, un algoritmo de *machine Learning* es aquel que es capaz de aprender a partir de datos. Estos algoritmos se pueden categorizar según el tipo de tarea, como clasificación, regresión, entre otros, o según su tipo de aprendizaje, ya sea supervisado o no supervisado [Goodfellow et al., 2016]. En el contexto de la clasificación, la tarea implica especificar a qué categoría pertenece una entrada dentro de un conjunto predefinido. Por otro lado, un algoritmo de aprendizaje supervisado se compone de un conjunto de datos que contienen características, donde cada ejemplo tiene asociado una etiqueta. El modelo aprende a hacer predicciones o clasificaciones basadas en la relación entre las entradas y las etiquetas proporcionadas durante el entrenamiento.

Dentro de machine learning, las redes neuronales [Goodfellow et al., 2016] han emergido como un pilar fundamental, transformando la manera en que abordamos problemas complejos en diversos dominios. Inspiradas en el funcionamiento del cerebro humano, estas estructuras computacionales han demostrado una extraordinaria capacidad para aprender patrones, representar información compleja y realizar tareas que van desde la clasificación de imágenes hasta la traducción de lenguajes.

2.7.1. RNN (Recurrent neural network)

Las Redes Neuronales Recurrentes (RNN) [Goodfellow *et al.*, 2016], a diferencia de las redes neuronales tradicionales, pueden recordar información de observaciones anteriores, lo que las hace adecuadas para problemas que involucran dependencias temporales.

A pesar de su capacidad para manejar secuencias temporales, las RNN convencionales pueden experimentar problemas de desvanecimiento del gradiente, lo que dificulta la retención de información en situaciones donde las dependencias temporales son significativas. Este fenómeno se debe a la propagación del error a través de múltiples pasos temporales, lo que resulta en la atenuación del gradiente y, por ende, en la pérdida de información importante.

2.7.2. LSTM (Long Short-Term Memory)

Las Redes de Memoria a Largo Plazo (LSTM) [Hochreiter y Schmidhuber, 1997] representan una evolución significativa en el campo de las RNN, abordando las limitaciones que estas últimas enfrentan en la retención de información a lo largo del tiempo. Estas han demostrado ser esenciales para el aprendizaje efectivo de dependencias a largo plazo en secuencias temporales.

La celda LSTM es la unidad fundamental que compone esta arquitectura. A diferencia de las unidades en RNN estándar, la celda LSTM incorpora tres puertas distintas: de entrada, de salida y de olvido.

Puerta de entrada: Controla qué nueva información se debe almacenar en la celda

Puerta de olvido: Decide qué información en la celda actual debe ser descartada.

Puerta de salida: Regula qué información de la celda se enviará como salida al siguiente paso temporal.

El diseño modular de las LSTMs permite que las celdas interactúen entre sí y mantengan una memoria a largo plazo eficiente.

2.7.3. Ensemble Learning

El Ensemble Learning [Hastie *et al.*, 2009], es un enfoque en el campo del aprendizaje automático que implica la combinación de múltiples modelos. La idea fundamental es que la combinación de varios modelos, cada uno con sus propias fortalezas y debilidades, puede resultar en un modelo conjunto más robusto y preciso.

Los métodos de Ensemble Learning se pueden implementar de diversas maneras, siendo dos de las estrategias más comunes:

Bagging: En este enfoque, se entrena cada modelo del conjunto de manera independiente.

Boosting: En lugar de entrenar modelos independientes, este enfoque se centra en mejorar iterativamente un único modelo débil. En cada iteración, el modelo se ajusta para corregir los errores cometidos en las predicciones anteriores.

2.7.4. Árbol de decisión

Un Árbol de Decisión [Breiman, 1984] es una estructura jerárquica en forma de árbol utilizada en el campo del aprendizaje automático y la estadística para tomar decisiones basadas en condiciones lógicas sobre un conjunto de características. Esta técnica se utiliza tanto para problemas de clasificación como de regresión, siendo un método eficaz y fácilmente interpretable.

El proceso de construcción de un árbol de decisión implica seleccionar iterativamente la característica que mejor divide el conjunto de datos en cada nodo, de acuerdo con algún criterio de impureza, como la ganancia de información o la reducción de la varianza.

Los árboles de decisión ofrecen ventajas notables, como la capacidad de manejar datos categóricos y numéricos, así como su interpretabilidad visual.

2.7.5. Random Forest Classifier

Random Forest [Breiman, 2001] es un algoritmo de *ensemble learning*, el cual combina múltiples árboles de decisión, para realizar predicciones. Cada árbol toma una decisión y, finalmente, la predicción del conjunto se determina mediante votación o promedio.

En cada paso de construcción de un árbol, el algoritmo selecciona un subconjunto aleatorio de características. Esto introduce variabilidad y evita la dependencia excesiva de ciertas características dominantes. Debido a esto *Random Forest* tiende a ser más resistente al *overfitting* que un solo árbol de decisión.

2.7.6. Normalización Min-Max

La Normalización Min-Max [Patro y Sahu, 2015] es una técnica de preprocesamiento en el campo de la estadística utilizada para escalar y ajustar los valores de una variable a un rango

específico. Su objetivo principal es transformar los datos de manera que todos los valores se ubiquen dentro de un intervalo predeterminado, típicamente entre 0 y 1.

El proceso de Normalización Min-Max garantiza que los datos transformados estén acotados dentro del rango especificado, lo que puede ser beneficioso para algoritmos de aprendizaje automático que son sensibles a las diferencias de escala entre las características. Algunos de los algoritmos que suelen beneficiarse de la normalización incluyen la regresión lineal y las redes neuronales.

CAPÍTULO 3

ESTADO DEL ARTE

3.1. Selección de Algoritmos en solvers basados en CDCL

En este capítulo se hará una revisión del trabajo relacionado sobre selección de algoritmos para el problema de Satisfacibilidad de fórmulas de lógica proposicional (SAT). PBO puede ser considerado como una generalización de SAT.

3.1.1. SATzilla

SATzilla es un método automatizado diseñado para construir portafolios de algoritmos por instancias, para el problema de satisfacibilidad booleana (SAT). Este enfoque se basa en 84 características para representar las instancias inicialmente propuestas por [Nudelman *et al.*, 2004], las cuales han sido modificadas para reducir el tiempo de cómputo, resultando en un conjunto final de 48 características que se muestran en la Figura 6.

Problem Size Features:

1. **Number of clauses:** denoted c
2. **Number of variables:** denoted v
3. **Ratio:** c/v

Variable-Clause Graph Features:

- 4-8. **Variable nodes degree statistics:** mean, variation coefficient, min, max and entropy.
- 9-13. **Clause nodes degree statistics:** mean, variation coefficient, min, max and entropy.

Variable Graph Features:

- 14-17. **Nodes degree statistics:** mean, variation coefficient, min and max.

Balance Features:

- 18-20. **Ratio of positive and negative literals in each clause:** mean, variation coefficient and entropy.
- 21-25. **Ratio of positive and negative occurrences of each variable:** mean, variation coefficient, min, max and entropy.
- 26-27. **Fraction of binary and ternary clauses**

Proximity to Horn Formula:

28. **Fraction of Horn clauses**
- 29-33. **Number of occurrences in a Horn clause for each variable:** mean, variation coefficient, min, max and entropy.

DPLL Probing Features:

- 34-38. **Number of unit propagations:** computed at depths 1, 4, 16, 64 and 256.
- 39-40. **Search space size estimate:** mean depth to contradiction, estimate of the log of number of nodes.

Local Search Probing Features:

- 41-44. **Number of steps to the best local minimum in a run:** mean, median, 10th and 90th percentiles for SAPS.
45. **Average improvement to best in a run:** mean improvement per step to best solution for SAPS.
- 46-47. **Fraction of improvement due to first local minimum:** mean for SAPS and GSAT.
48. **Coefficient of variation of the number of unsatisfied clauses in each local minimum:** mean over all runs for SAPS.

Figura 6: Características utilizadas para construir *SATzilla07* Fuente: [Xu *et al.*, 2008]

Estas características se dividen en dos categorías principales según su naturaleza. En primer lugar, se encuentran las **características estáticas**, por ejemplo, aquellas directamente relacionadas con la estructura del problema, como la cantidad de cláusulas o los estadísticos

asociados al grafo de variables y cláusulas. Estas características son relativamente más simples de calcular y ofrecen una visión estática del problema.

En contraste, las **probing features** constituyen la segunda categoría y se obtienen durante la ejecución del algoritmo. Por ejemplo, pueden incluir información sobre la cantidad de propagaciones unitarias que se producen durante la búsqueda. Estas características, aunque proporcionan información valiosa en tiempo real, tienden a ser más costosas de calcular debido a su dependencia del proceso dinámico de resolución del problema.

La eficacia de estas características fue comprobada en la competición SAT del 2007, donde los solvers *SATzilla07* obtuvieron tres medallas de oro, una de plata y una de bronce, destacando su excelente rendimiento [Xu *et al.*, 2008].

Cabe destacar que si bien SAT y PBO están relacionados, no existe una traducción de PBO a SAT que sea polinomial y arco-consistente y, por tanto, estas características no pueden ser extraídas de forma sencilla para instancias PBO.

3.1.2. SATfeatPy

SATfeatPy introduce una librería que proporciona técnicas de extracción de características diseñadas para problemas SAT en formato CNF [Provan-Bessell *et al.*, 2022]. Las características extraídas corresponden a las propuestas por *SATzilla*, se dividen en *SATzilla base* que contiene solo las características estáticas y *SATzilla full* que contiene el conjunto completo de características. Se compara con otras propuestas más actuales como ANT [Ansótegui *et al.*, 2017] y ALF [Alfonso y Manthey, 2018], además del conjunto de todas las características juntas, denominado *all*.

Para comparar las caracterizaciones se proponen dos problemas. En primer lugar, el problema de satisfacibilidad, este corresponde a una tarea de clasificación binaria, donde se busca predecir si una instancia es satisfacible o no. El segundo problema a considerar es el de clasificación de instancias en problemas, este se refiere a identificar el dominio al que pertenece cada instancia y corresponde a una tarea de clasificación multi-clases. En ambos casos, se construyeron modelos de clasificación de *Random Forest* y *XGBoost* [Chen y Guestrin, 2016]. Además, el *dataset* utilizado está compuesto por 3000 instancias en forma CNF, con la misma cantidad de instancias satisfacibles e insatisfacibles.

Para el primer problema, los resultados se muestran en la tabla 1, donde se puede notar que *SATzilla full* obtiene el mejor rendimiento respecto a las otras caracterizaciones, con un 99,45 %, sin embargo, es también el más costoso en términos de computo de características.

Feature Set	Random Forest (RFC)	XGBoost (XGB)	Computational Time (s)
<i>SATzilla base</i>	91.33 %	90.68 %	0.03
<i>SATzilla full</i>	99.45 %	99.54 %	3.18
<i>ANT</i>	85.94 %	81.82 %	0.15
<i>ALF</i>	91.65 %	93.85 %	119.54
<i>All</i>	98.93 %	99.54 %	122.89

Tabla 1: Valores del *accuracy* y tiempo promedio computacional (en segundos) para el problema de satisfacibilidad sobre los conjuntos de características Fuente: [Provan-Bessell et al., 2022]

Por otro lado para el caso de clasificación de instancias en categorías de problemas se muestran los resultados en la tabla 2, se puede notar que ahora el conjunto de todas las características juntas fue el mejor, aun así las propuestas de *SATzilla base*, *SATzilla full* y *ALF* lograron valores de *accuracy* sobre el 99 %.

Feature Set	Random Forest (RFC)	XGBoost (XGB)
<i>SATzilla base</i>	99.65 %	99.66 %
<i>SATzilla full</i>	99.59 %	99.74 %
<i>ANT</i>	93.14 %	93.47 %
<i>ALF</i>	99.79 %	99.64 %
<i>All</i>	99.83 %	99.81 %

Tabla 2: Valores del *accuracy* y tiempo promedio computacional (en segundos) para el problema de categorización en familias de problemas sobre los conjuntos de características Fuente: [Provan-Bessell et al., 2022]

Estos resultado sugieren que las caracterizaciones utilizadas ayudan a clasificar correctamente las instancias en familias representando la estructura del problema más que su satisfacibilidad.

3.1.3. LMPick

LMPick es un selector de estrategia de reinicio por instancias, propuesto por [Haim y Walsh, 2009], para *SAT-solvers* basados en CDCL. El portafolio utilizado comprende cuatro técnicas de reinicio y sus variaciones de parámetros, estos se ven reflejados en un total de 9 algoritmos.

La caracterización propuesta para cada instancia se basa en cuatro conjuntos inspirados en las técnicas de *SATzilla*, capturando características en distintos momentos de la ejecución del *solver*. La figura 7 muestra el listado de características utilizadas, estas se clasifican en 4 conjuntos:

Conjunto I: Este solo contiene la cantidad de variables y cláusulas antes de comenzar la búsqueda.

Conjunto II: Este también se obtiene antes de que comience la búsqueda, pero después del proceso de remover las cláusulas que ya se satisfacen, reducción de cláusulas y propagación unitaria de la formula original.

Conjunto III: Este incluye estadísticas recolectadas durante una ventana de observación de 2000 conflictos.

Conjunto IV: El ultimo conjunto esta compuesto de las misma características anteriores, pero se obtienen después de la ventana de observación.

Se utilizan ventanas de observación debido a que los primeros reinicios suelen tener mucho ruido, por lo que se obtiene mejores resultados al recolectar los datos con la búsqueda ya avanzada

Set I:

1. Number of variable
2. Number of clauses

Set II:

3. Number of variable
4. Number of clauses
5. Variable to clauses ratio
6. Number of binary clauses
7. Number of ternary clauses
8. Number of horn clauses
- 9-12. VCG - Variable nodes degree statistics: mean, variation coefficient, min and max
- 13-16. VCG - Clause nodes degree statistics: mean, variation coefficient, min and max
- 17-20. Number of occurrences in Horn clauses: mean, variation coefficient, min and max
- 21-24. Ratio of positive and negative occurrences of variables: mean, variation coefficient, min and max
25. Number of assigned variable

Set III:

- 26-29. Backjump Size: mean, variation coefficient, min and max
- 30-33. Search Depth: mean, variation coefficient, min and max
- 34-37. log(WBE) value : mean, variation coefficient, min and max

Set IV:

38. Number of variable
39. Number of clauses
40. Variable to clauses ratio
41. Number of binary clauses
42. Number of ternary clauses
43. Number of horn clauses
- 44-47. VCG - Variable nodes degree statistics: mean, variation coefficient, min and max
- 48-51. VCG - Clause nodes degree statistics: mean, variation coefficient, min and max
- 52-55. Number of occurrences in Horn clauses: mean, variation coefficient, min and max
- 56-59. Ratio of positive and negative occurrences of variables: mean, variation coefficient, min and max
60. Number of assigned variable

Figura 7: Características utilizadas para construir *LMPick* Fuente: [Haim y Walsh, 2009]

Basándose en la observación de que las instancias satisfacibles y las insatisfacibles presentan distribuciones de tiempos de ejecución distintos, se procedió a entrenar un modelo de clasificación de satisfacibilidad, seguido de esto, se entrenaron dos modelos de regresión por cada uno de los algoritmos del portafolio, los cuales predicen el tiempo de ejecución específico de cada algoritmo.

Para la construcción del modelo de clasificación, se optó por la técnica de regresión logística. En cambio, para los modelos del tiempo de ejecución, se utilizó la regresión lineal *Ridge*.

Los resultados revelan un rendimiento positivo en los casos de las instancias satisfacibles, superando a un selector aleatorio y mostrando un rendimiento igual o superior al *Single Best Solver*. Sin embargo, en el caso de instancias insatisfacibles, se observa un rendimiento inferior, por lo que el selector no está aprendiendo la mejor técnica de reinicio.

3.2. Machine Learning Restarts

El *solver Glucose* introdujo el concepto de LBD, demostrando su eficacia para mejorar la búsqueda en los *solvers* basados en CDCL. Otro enfoque relacionado se presenta a través de una nueva política de reinicio que utiliza el historial de LBDs de las instancias para el problema SAT [Jia *et al.*, 2018].

El análisis se centra en los valores LBDs recopilados de 350 instancias provenientes de la competición SAT del 2017. Los resultados indican que los reinicios mejoran la calidad de las cláusulas aprendidas, y reiniciar con frecuencia disminuye los valores de LBD de las cláusulas aprendidas, mejorando así el rendimiento del *solver*. Sin embargo, un exceso de reinicios puede ser contraproducente debido al costo computacional de reconstruir el árbol de búsqueda muy frecuentemente.

En respuesta a este desafío, proponen una política de reinicio basada en *machine learning* (MLR), que predice la calidad de la siguiente cláusula a aprender basándose en el historial previo de LBDs. Se reinicia si la calidad de la cláusula predicha cae por debajo de un límite dinámico determinado en base al historial del *solver*. Este límite se define en función del percentil 99.9 de LBDs de las cláusulas aprendidas durante la ejecución. Según los resultados experimentales, los valores de LBD exhiben una distribución normal.

Para la predicción del valor de LBD futuro, se entrena un modelo de regresión lineal utilizando el algoritmo Adam [Kingma y Ba, 2014] para ajustar los parámetros en cada iteración. Los coeficientes se inicializan en 0 al comienzo de la búsqueda y se actualizan con cada cláusula aprendida mediante una iteración de Adam.

La evaluación experimental se lleva a cabo utilizando instancias recopiladas de las competiciones SAT desde 2014 hasta 2017. La Figura 8 ilustra el rendimiento del *solver*, con un límite de tiempo de 5000 segundos, demostrando ser competitivo con el estado del arte, ubicándose entre las técnicas Luby y *Glucose*. Una contribución destacada radica en la aplicación de modelos de *machine learning* para predecir los valores futuros de LBDs, mostrando así la capacidad de la política MLR para mejorar el rendimiento del *solver*.

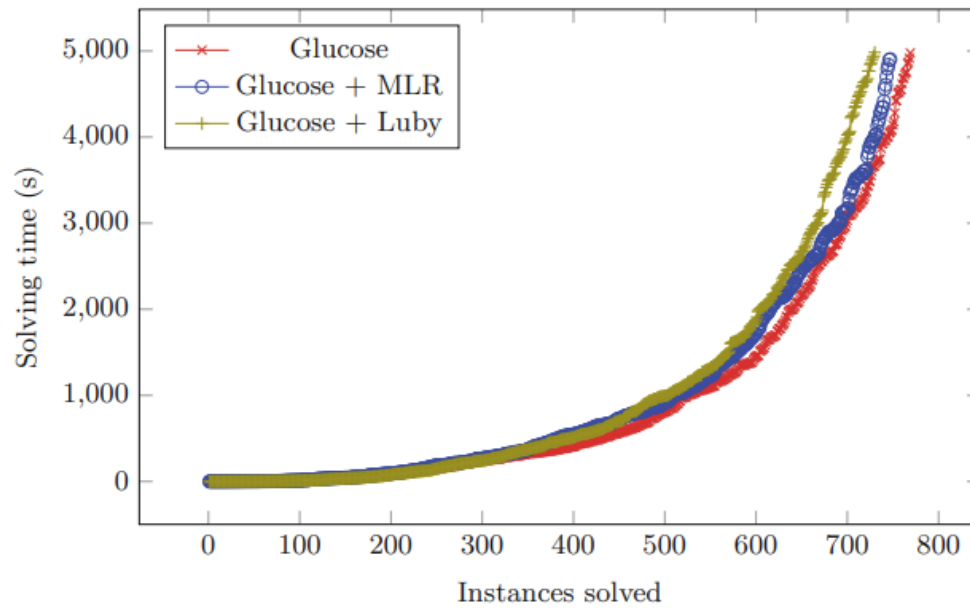


Figura 8: *Cactus-plot* de las políticas de reinicio *Luby*, *Glucose* Y *MLR*. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: [Jia et al., 2018]

CAPÍTULO 4

PROPUESTA DE SOLUCIÓN

Se presentan los detalles del desarrollo de la solución. Para ello, la figura 9 proporciona una visión general de los pasos implicados en el proceso de la construcción del algoritmo selector. A continuación, se desglosarán cada una de las tareas

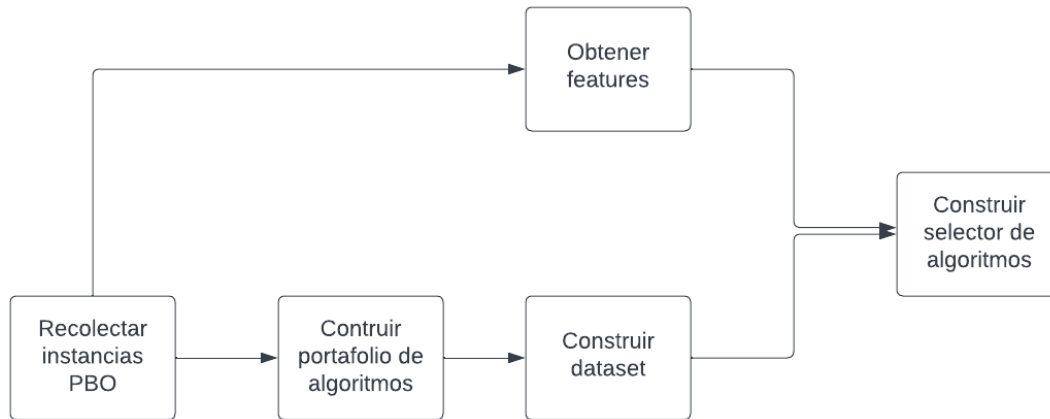


Figura 9: Esquema del flujo de trabajo Fuente: Elaboración propia

4.1. Tecnologías

Para la experimentación se utilizó el cluster Savio de la Universidad de California, Berkeley. En concreto, se utilizaron dos particiones con la siguiente configuración:

1. savio 1: Modelo CPU Intel Xeon E5-2670 v2, con 64GB de memoria y 20 núcleos
2. savio 2: Modelo CPU Intel Xeon E5-2670 v3, con 64 GB de memoria y 24 núcleos.

Para la implementación de los modelos se utilizó *Python* y la librería *Keras*.

4.2. Instancias PBO

Se utilizó el *benchmark* de la competición *Pseudo-Boolean 2016*, la cual comprende instancias PBO obtenidas de las competiciones: 2006, 2007, 2009, 2010, 2011, 2012, 2015 y 2016.

Las cuales provienen de diferentes dominios, entre ellas se encuentran *Course Assignment*, *Routing*, *Traveling Salesperson Problem*.

Además, las instancias se encuentran organizadas en carpetas o familias, cada una contiene problemas con características similares. Estas se pueden clasificar de la siguiente manera:

1. Según su función objetivo

- a) DEC: No hay función objetivo a optimizar, es un problema decisional.
- b) OPT: Hay una función objetivo a optimizar, el *solver* debe encontrar la solución con el mejor valor posible.

2. Según el tamaño de sus coeficientes

- a) SMALLINT: no hay restricciones donde la suma de los coeficientes sea mayor a 2^{20} .
- b) BIGINT: al menos una restricción tiene una suma de coeficientes mayor a 2^{20} .

3. Según la linealidad de sus restricciones

- a) LIN: todas las restricciones son lineales
- b) NLC: al menos una de sus restricciones es no lineal.

El total de instancias son 5467, de las cuales 2324 corresponden a problemas decisionales y 3143 a problemas de optimización. Por otro lado, las instancias están separadas en 219 familias de problemas, donde el problema *Proof Complexity* en la categoría decisional tiene la mayor cantidad de instancias. Para más detalles respecto la cantidad de instancias por familia ver *anexo 1*

4.3. Construcción del portafolio

Para construir el portafolio de algoritmos se seleccionaron las técnicas de reinicio *Luby* y *Glucose*. En este proceso, se implementó la técnica de reinicio de *Glucose* en el *solver Rounding-SAT*, mientras que la técnica *Luby* se mantuvo como configuración predeterminada. Posteriormente, se llevaron a cabo experimentos sobre los parámetros: *luby base*, *luby multiplier*, *glucose window* y *glucose multiplier*.

Se realizaron experimentos con el fin de seleccionar los parámetros más óptimos para cada técnica. Para ello, se eligieron dos conjuntos de instancias acotadas, enfocándonos en aquellas con mayores tiempos de ejecución para cada técnica de reinicio. En particular, para la técnica *Luby*, se trabajó con un conjunto de 128 instancias, mientras que para *Glucose* se consideraron 192 instancias. Cabe destacar que todas las instancias fueron ejecutadas con

un límite de tiempo de resolución establecido en 3600 segundos. Además, los experimentos en *Glucose* fueron ejecutados en la partición *savio 1*, mientras que para *Luby* se utilizó *savio 2*. Es relevante destacar que cada técnica fue analizada de manera independiente, sin realizar comparaciones directas entre los resultados obtenidos por cada una.

Las tablas 3 y 4 definen los experimentos realizados y la configuración de parámetros para las técnicas de reinicio *Luby* y *Glucose* respectivamente.

Nombre	<i>Luby Base</i>	<i>Luby multiplier</i>
luby_100_2	100	2
luby_100_4	100	4
luby_100_6	100	6
luby_200_2	200	2
luby_500_2	500	2
luby_1000_2	1000	2

Tabla 3: Configuración de parámetros de los experimentos realizados con la técnica de reinicio *Luby* Fuente: Elaboración Propia.

Nombre	Glucose window	Glucose multiplier
exp_50_5	50	0.5
exp_50_7	50	0.7
exp_50_8	50	0.8
exp_50_9	50	0.9
exp_100_5	100	0.5
exp_100_7	100	0.7
exp_100_8	100	0.8
exp_100_9	100	0.9
exp_200_5	200	0.5
exp_200_7	200	0.7
exp_200_8	200	0.8
exp_200_9	200	0.9
exp_400_5	400	0.5
exp_400_7	400	0.7
exp_400_8	400	0.8
exp_400_9	400	0.9
exp_500_5	500	0.5
exp_500_7	500	0.7
exp_500_8	500	0.8
exp_500_9	500	0.9
exp_1000_5	1000	0.5
exp_1000_7	1000	0.7
exp_1000_8	1000	0.8
exp_1000_9	1000	0.9

Tabla 4: Configuración de parámetros en los experimentos realizados con la técnica de reinicio *Glucose* Fuente: Elaboración Propia.

En relación con los resultados obtenidos, se llevó a cabo un análisis detallado que abarcó la cantidad de instancias resueltas, el tiempo de ejecución y la complementariedad entre cada configuración. Es importante destacar que, en este último aspecto, se consideraron mejores aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras significativas, alcanzando una reducción del 40% en el tiempo de resolución.

Las tablas 5 y 6 muestran la cantidad de instancias resueltas por cada configuración de parámetros para las técnicas de *Glucose* y *Luby* respectivamente. En el primero destaca el experimento *exp_100_7* con 163 instancias resueltas. Por otro lado, en *Luby*, el experimento *luby_100_2* logra un total de 122 instancias resueltas.

Nombre	Total de instancias resueltas
exp_50_5	155
exp_50_7	155
exp_50_8	147
exp_50_9	140
exp_100_5	149
exp_100_7	163
exp_100_8	151
exp_100_9	148
exp_200_5	147
exp_200_7	153
exp_200_8	155
exp_200_9	151
exp_400_5	145
exp_400_7	155
exp_400_8	159
exp_400_9	155
exp_500_5	150
exp_500_7	153
exp_500_8	155
exp_500_9	155
exp_1000_5	147
exp_1000_7	152
exp_1000_8	149
exp_1000_9	151

Tabla 5: Cantidad de instancias resueltas por cada configuración de parámetros de *Glucose*
Fuente: Elaboración Propia.

Nombre	Total de instancias resueltas
luby_100_2	122
luby_100_4	100
luby_100_6	99
luby_200_2	101
luby_500_2	102
luby_1000_2	104

Tabla 6: Cantidad de instancias resueltas por cada configuración de parámetros de *Luby* Fuente: Elaboración Propia.

Las figuras 11 y 10 representan el *cactus-plot* para los experimentos en las técnicas de reinicio *Glucose* y *Luby* respectivamente. Este tipo de gráfico permite visualizar la cantidad de instancias resueltas y la velocidad en que las resuelve.

Para el caso de *Glucose*, la configuración *exp_100_7* es la que resuelve mas instancias, y es la segunda mas rápida de todas. Por otro lado, en el caso de *Luby*, se confirma que la configuración *luby_100_2* es la que resuelve mas instancias, sin embargo no es la mas rápida. Entre las otras destaca *luby_100_4*, siendo la mas rápida de todas.

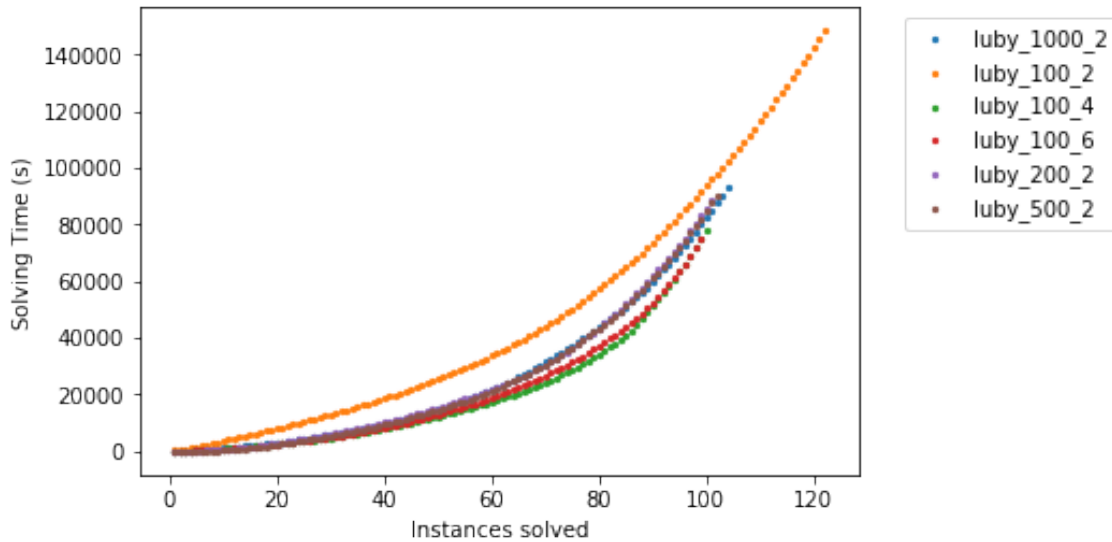


Figura 10: *Cactus-plot* de las configuraciones de parámetros para la técnica de reinicio *Luby*. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia

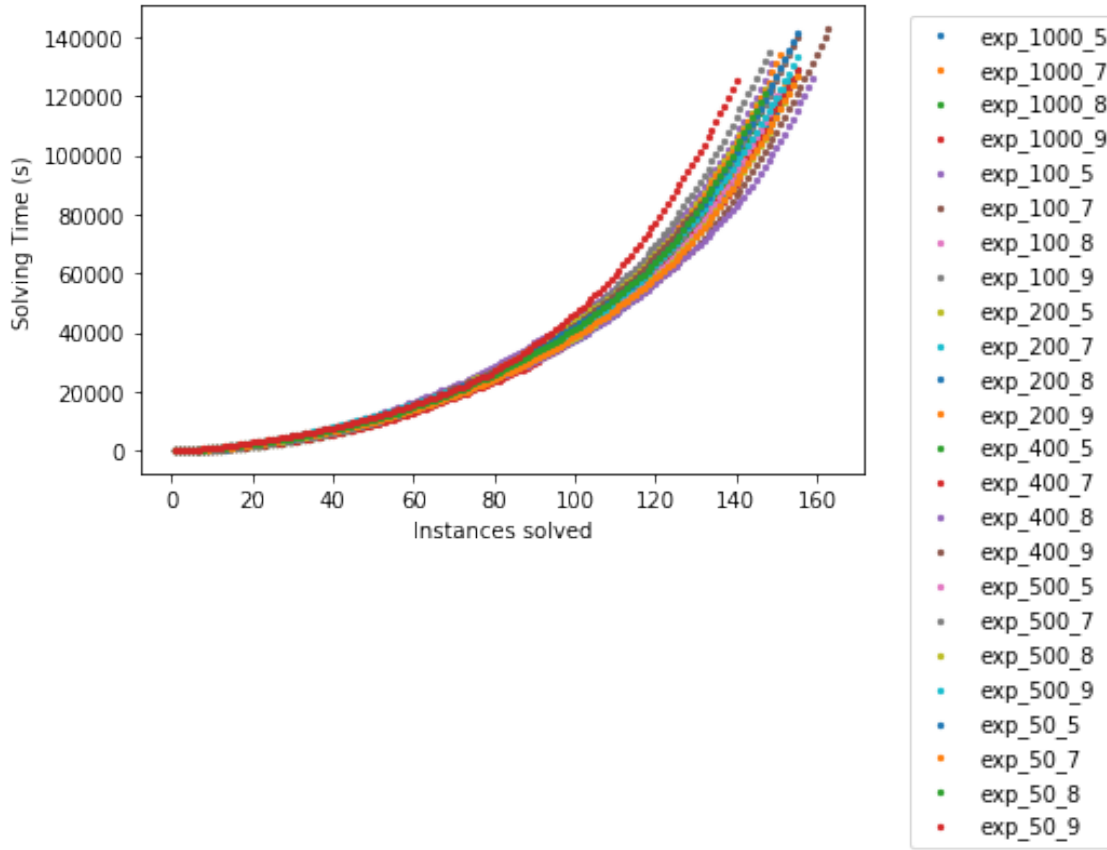


Figura 11: *Cactus-plot* de las configuraciones de parámetros para la técnica de reinicio *Glucose*. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia

Las figuras 13 y 12 representan la matriz de complementariedad sobre cada par de configuraciones para las técnicas *Glucose* y *Luby* respectivamente. En el caso de *Glucose*, se puede notar que la configuración más complementaria a *exp_100_7* es *exp_50_9*. Por otro lado, en *Luby* la configuración más complementaria a *luby_100_2* es *luby_100_4*. Por lo tanto, el portafolio de algoritmos elegidos para construir el selector lo componen: *exp_100_7*, *exp_50_9*, *luby_100_2* y *luby_100_4*

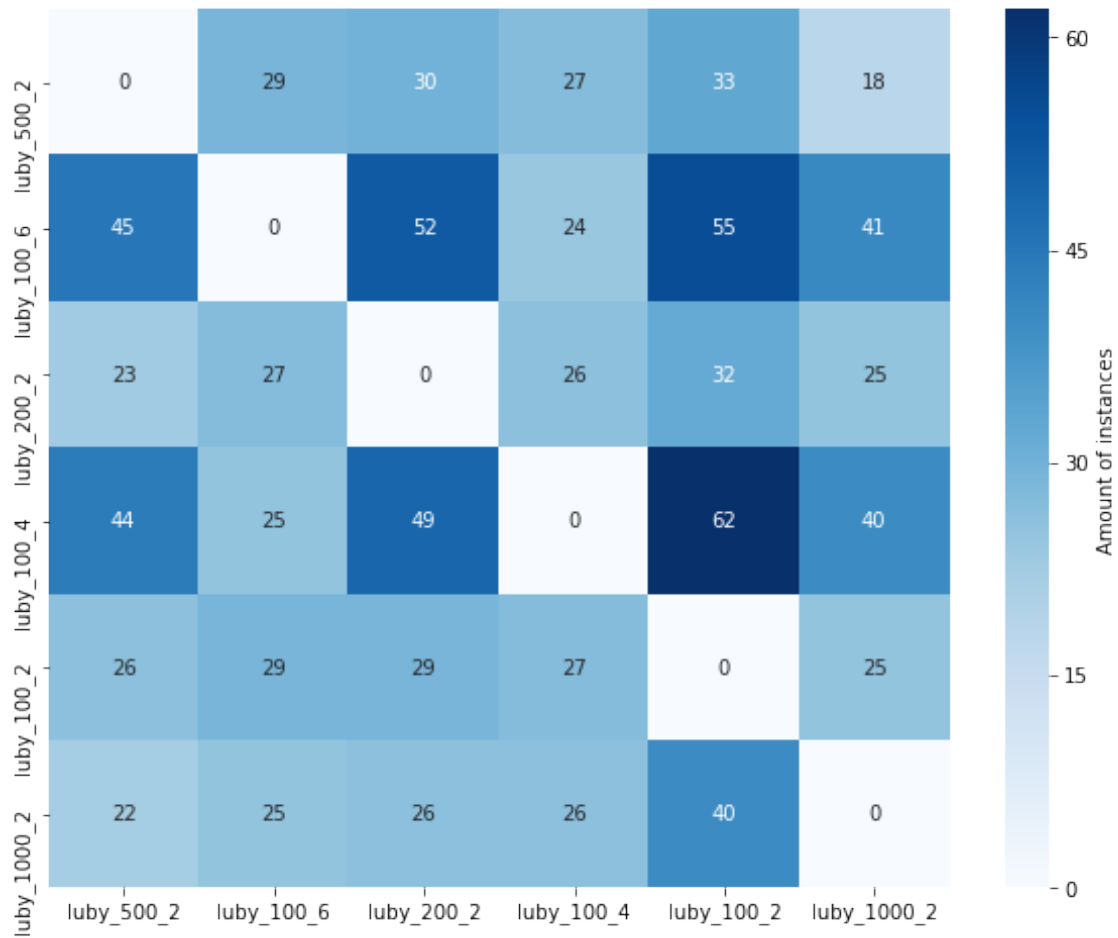


Figura 12: Matriz de complementariedad entre cada par de configuraciones de *Luby*. Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia

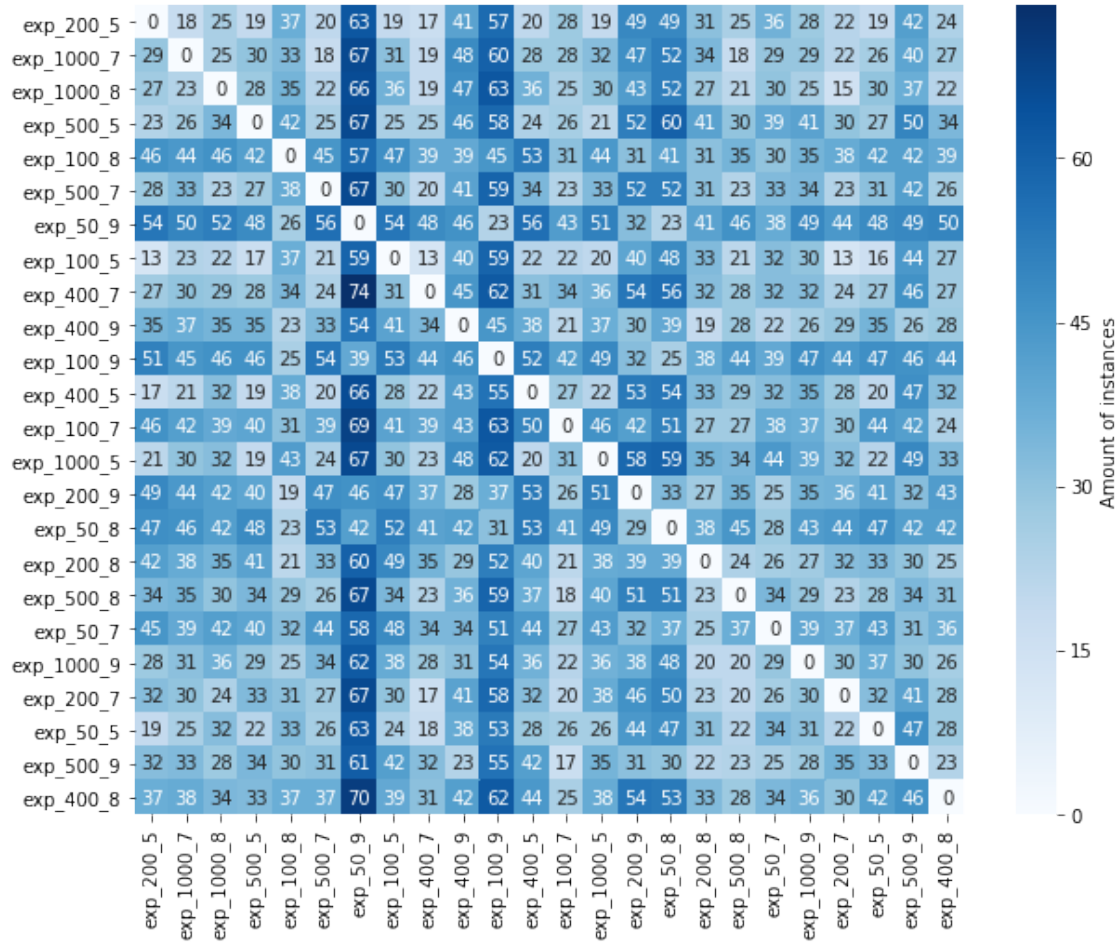


Figura 13: Matriz de complementariedad entre cada par de configuraciones de *Glucose*. Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia

4.4. Construcción del *dataset*

Sobre el portafolio de algoritmos definidos en la sección anterior se obtiene el rendimiento de cada uno sobre el conjunto total de instancias. Para esto se realiza un experimento en la partición *savio 2*, con un límite de tiempo definido en 3600 segundos.

Luego, analizando los resultados de la tabla 7 se observa que el algoritmo del portafolio con mayor cantidad de instancias resueltas es *luby_100_2*, que corresponde a la configuración por defecto de *RoundingsAT*. Además, el VBS puede resolver 3829 instancias, lo que corresponde a una mejora del 2 %. Aunque este aumento no sea considerable, resulta relevante señalar que una proporción considerable de las instancias son fáciles de resolver, como se

puede apreciar en la figura 14, donde la curva comienza a mostrar un aumento pronunciado a partir de las 3000 instancias.

Nombre	Total de instancias resueltas
luby_100_2	3729
luby_100_4	3741
exp_100_7	3722
exp_50_9	3724
VBS	3829

Tabla 7: Cantidad de instancias resueltas por cada algoritmo del portafolio, incluye el *Virtual Best Solver* Fuente: Elaboración Propia.

Para respaldar lo mencionado, en la figura 14 se observan diferencias pequeñas de los tiempos y cantidad de instancias resueltas entre los cuatro algoritmos, sin embargo, el VBS es notablemente mejor en comparación. Finalmente, la figura 15 muestra la alta complementariedad entre los algoritmos elegidos para el portafolio.

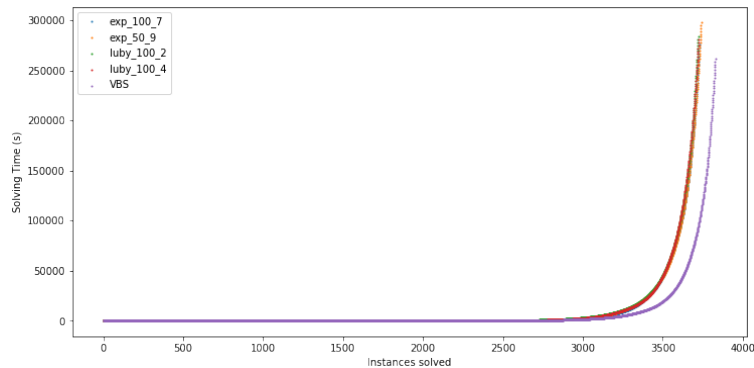


Figura 14: Cactus-plot de los algoritmos del portafolio. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia

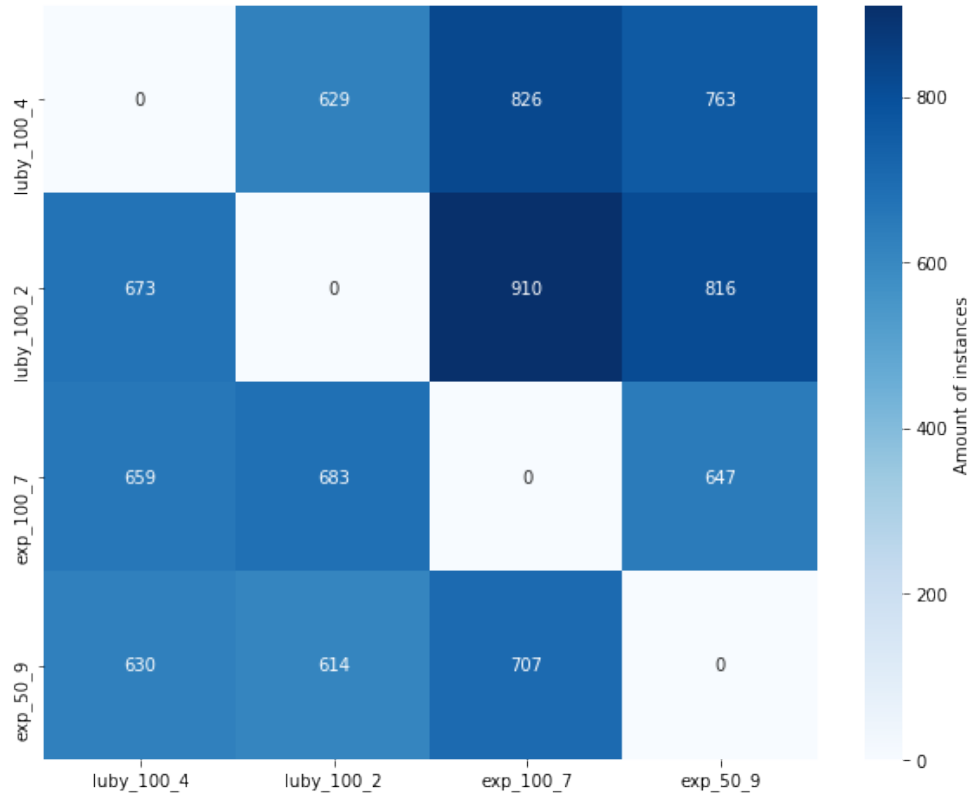


Figura 15: Matriz de complementariedad entre cada par de algoritmos del portafolio. Considera aquellas instancias que una configuración pudo resolver mientras que la otra no, así como aquellas que presentaron mejoras sobre el 40 % en el tiempo de resolución. Fuente: Elaboración propia

4.4.1. Objetivo a predecir

El objetivo del selector es predecir cual de los algoritmos tiene el mejor rendimiento en cada instancia. Esta tarea se aborda como un problema de clasificación multiclases, donde cada categoría representa uno de los algoritmos del portafolio. La figura 16 presenta el histograma con la frecuencia de cada categoría sobre el *dataset*. Se puede notar que las clases están balanceadas.

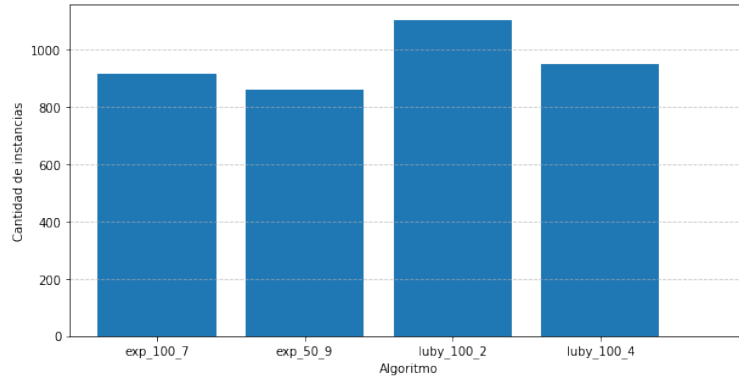


Figura 16: Histograma de las veces que un algoritmo gana sobre una instancia Fuente: Elaboración propia

Es relevante destacar que existe un conjunto significativo de instancias que pueden ser resueltas por todos los algoritmos en menos de 5 segundos, como se observa en la categoría “None” del histograma en la figura 17. Esto sugiere que, para muchas instancias, los rendimientos de los algoritmos son comparativamente similares, lo que podría limitar la capacidad del selector para discernir de manera significativa entre ellos. Es esencial reconocer que solo unas pocas instancias, donde las diferencias en el rendimiento son más pronunciadas, tendrán un impacto significativo en el selector.

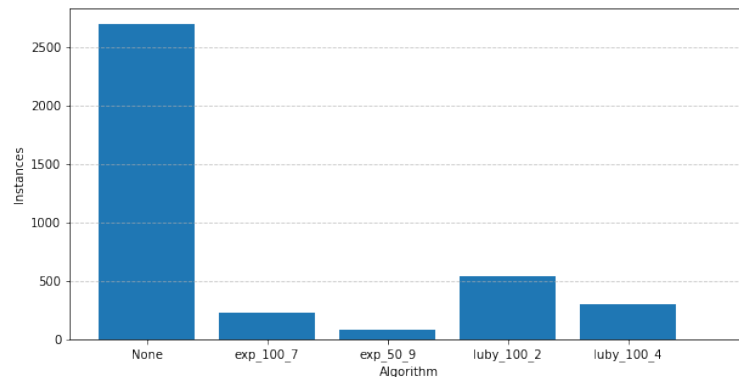


Figura 17: Histograma que refleja la frecuencia en que un algoritmo gana en una instancia, excluyendo aquellas en las que todos los algoritmos la resuelven en menos de 5 segundos. Estas ultimas se ven agrupadas en la categoría “None” Fuente: Elaboración propia

Por ultimo, se muestra en la figura 18 los algoritmos elegidos por cada familia de instancias, se puede notar que la selección de políticas de reinicio no está fuertemente correlacionada con la pertenencia de las instancias a una familia específica. Esto indica que identificar la familia de una instancia, no es información suficiente para seleccionar la mejor política de reinicio para resolver esa instancia.

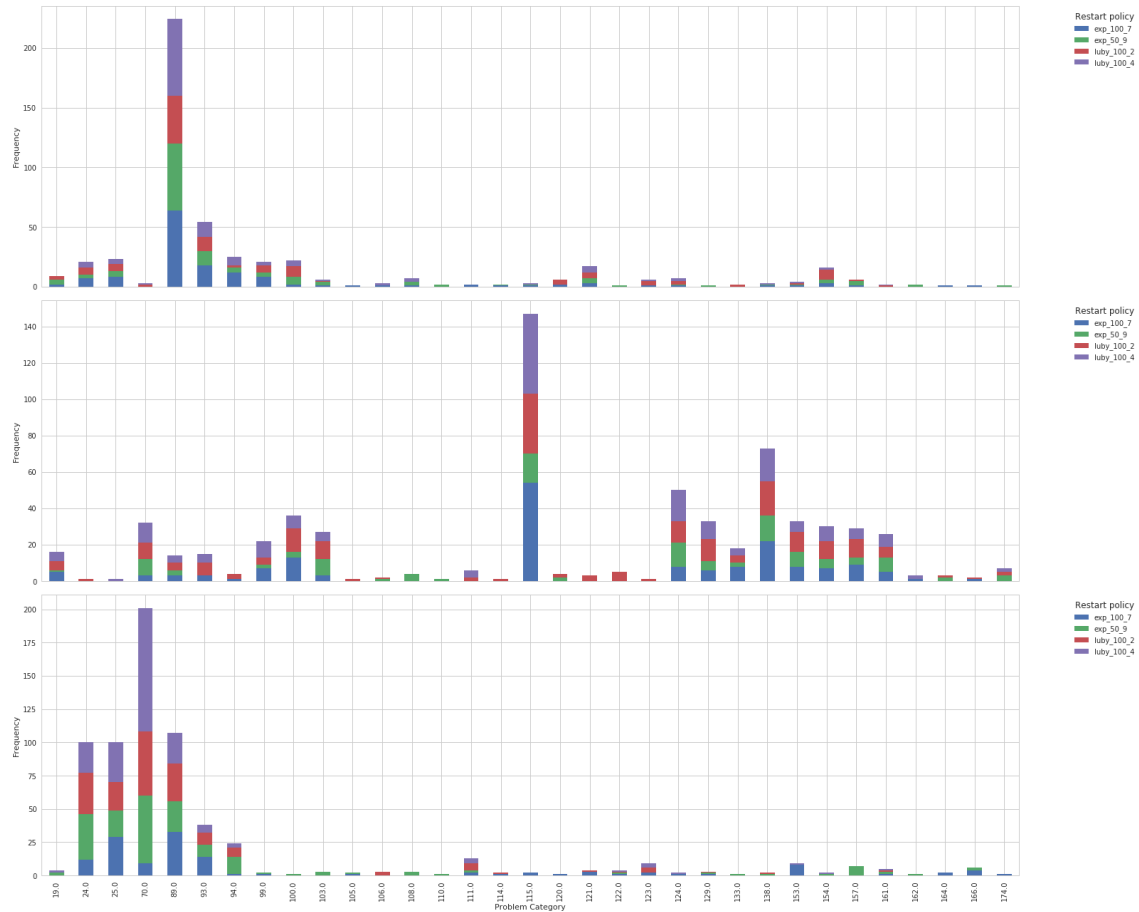


Figura 18: Histograma de la cantidad de instancias por familia, los colores representan el algoritmo que gana esa instancia Fuente: Elaboración propia

4.5. Obtención de *features*

En esta sección, se detallarán las dos caracterizaciones de instancias que serán evaluadas en el próximo capítulo. La primera corresponde al historial de LBD recolectados al comienzo de la búsqueda. Este enfoque, corresponde a una propuesta propia, inspirada en trabajos previos. La segunda se basa en las características propuestas por SATZilla [Xu *et al.*, 2008], obtenidas a través de herramientas como Naps [Sakai y Nabeshima, 2015] y SATfeatPy [Provan-Bessell *et al.*, 2022].

4.5.1. SATZilla

Para obtener las características propuestas por SATZilla, se utilizan dos herramientas, la primera es la librería SATfeatPY, que implementa técnicas de extracción de características para

problemas SAT en su forma CNF.

Para llevar a cabo la transformación de instancias PBO a su representación en forma CNF, se emplea el *solver* pseudo booleano denominado *NaPS* (*Nagoya pseudo-Boolean solver*) [Sakai y Nabeshima, 2015]. Este *solver*, fundamentado en *MiniSAT* [Eén y Sörensson, 2006], se utiliza específicamente para la conversión de instancias PBO a su equivalente en forma CNF. Durante este proceso, el algoritmo implementa la creación de nuevas variables, lo que resulta en un crecimiento exponencial en las instancias representadas en su formato CNF. Dada la complejidad inherente a esta tarea, se optará por limitar la obtención de características a aquellas básicas proporcionadas por *SATZilla*.

4.5.2. LBD

Para obtener el historial de LBD de cada instancia, se llevo a cabo un experimento en la partición *savio 2*, ejecutando el *solver* *RoundingSAT* sin ninguna técnica de reinicio para recolectar los primeros 200 valores de LBD. Dada la naturaleza aleatoria de los *solvers*, se optó por descartar los primeros 100 valores del inicio de la búsqueda, ya que podrían no estar vinculados con características intrínsecas de las instancias. Posteriormente, se emplearon los siguientes 100 valores como características para entrenar el selector de algoritmos.

Dado que las instancias más grandes tienden a tener valores más grandes de LBD, se realizó una normalización dividiendo los valores de LBD por la cantidad de variables de la instancia. Además, se ajustaron los valores que superaban el 70 % de la cantidad total de variables a 1. Esto se hizo para escalar los casos en los que la búsqueda a través del árbol es considerablemente extensa. En la figura 19, se muestra una serie de LBD y sus valores normalizados para una instancia específica. La gráfica refleja el comportamiento de los valores a lo largo de la ejecución, y al compararla con la curva de los valores normalizados, se aprecia cómo disminuye la escala de estos últimos. En esta instancia en particular, los valores de LBD durante la búsqueda alcanzan un poco más de 50.

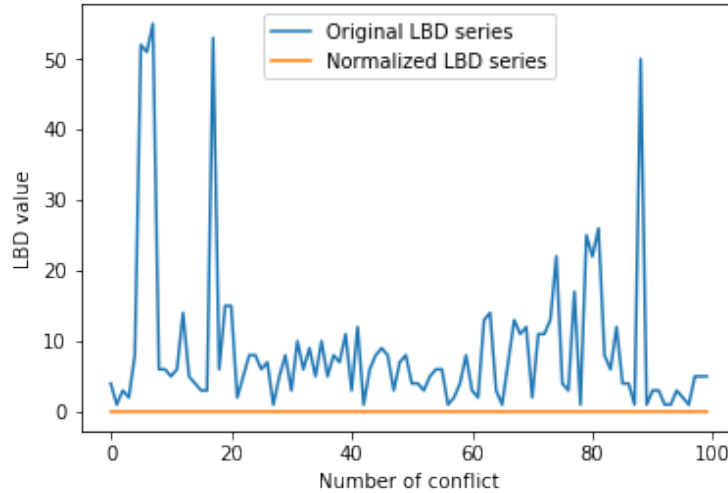


Figura 19: Gráfico de las curvas de valores de LBD originales y normalizados a lo largo de la búsqueda Fuente: Elaboración propia

4.6. Selector de Algoritmos

A continuación se entregan detalles de los selectores de algoritmos construidos para ambas caracterizaciones.

En el caso del historial de LBD, se utiliza un modelo de *LSTM*, para entrenar las secuencias de valores. La tabla 8 define las capas de la red neuronal y la cantidad de parámetros a entrenar. En la capa de salida, se emplea la función de activación *softmax* y la función de pérdida seleccionada es *categorical crossentropy* [Murphy, 2022], para la optimización se ha optado por el algoritmo Adam [Kingma y Ba, 2014].

Capa (Cantidad)	Cantidad de parametros
LSTM(100)	40800
Dense(10)	1010
Dense(4)	44

Tabla 8: Descripción de capas de la red neuronal creada con la cantidad de parámetros a entrenar por cada una Fuente: Elaboración propia

Para la caracterización de *SATZilla*, se utiliza la implementación del estado del arte [Provan-Bessell *et al.*, 2022], para esto se implementa un modelo *Random Forest Classifier*, con los siguientes parámetros:

1. `n_estimators = 10`

2. `criterion = gini`
3. `max_depth = sqrt`
4. `min_impurity_decrease=0.0`

CAPÍTULO 5

VALIDACIÓN DE LA SOLUCIÓN

En esta sección, se presentan los detalles de la metodología de validación y los resultados obtenidos.

5.1. Conjunto de validación

Como se discutió en la sección anterior, nos enfrentamos a un escenario donde muchas de las instancias son fáciles de resolver. Para evaluar de manera rigurosa el rendimiento del selector, se ha construido un conjunto de validación conformado por las instancias mas desafiantes en término de tiempo de resolución.

Otro factor a considerar a la hora de construir el conjunto de validación, es que a diferencia de los problemas de clasificación para SAT, identificar la familia de una instancia, no es información suficiente para seleccionar la mejor política de reinicio para resolver esa instancia. Por lo tanto, en la validación, no se consideró la pertenencia a una familia específica; en cambio, se aseguró simplemente la representación de una cantidad significativa de técnicas de reinicio, similares a las presentes en el conjunto de datos original.

Por ultimo, durante la recolección del historial de LBD no fue posible obtener los primeros 200 valores en todas las instancias. Como resultado, el conjunto de datos final esta compuesto de 2177 instancias. Posteriormente, se llevó a cabo una división de los datos en conjuntos de entrenamiento y validación, asignando el 75 % (1633 instancias) al conjunto de entrenamiento y el 25 % (544 instancias) al conjunto de validación. Los detalles de esta partición y la distribución de clases se pueden observar en las Figuras 20 y 21, que presentan los histogramas correspondientes a los datos de entrenamiento y validación, respectivamente.

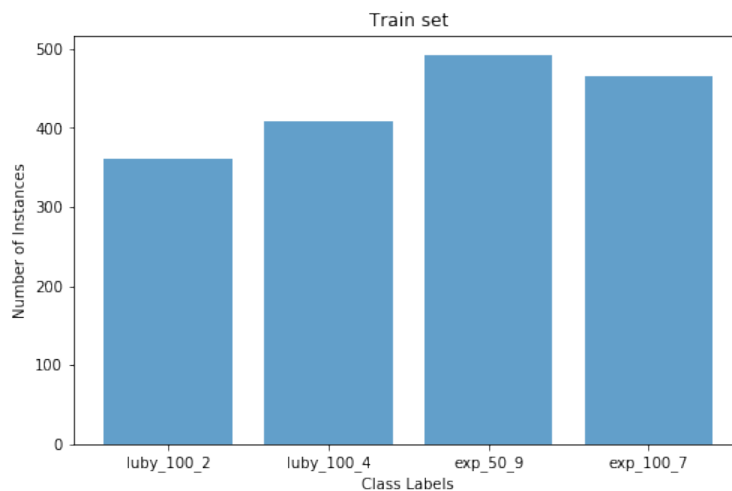


Figura 20: Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de entrenamiento Fuente: Elaboración propia

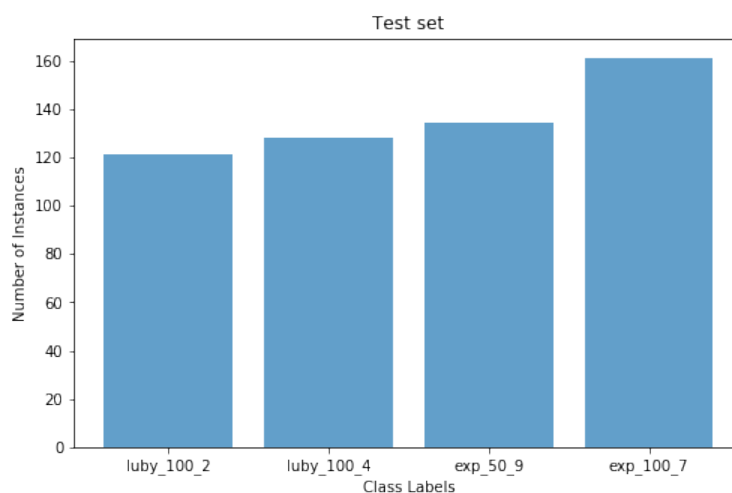


Figura 21: Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación Fuente: Elaboración propia

5.2. Procesamiento datos

En el procesamiento de las características de *SATzilla*, se siguió la misma implementación utilizada por el estado del arte [Provan-Bessell *et al.*, 2022], que incluye la normalización *min-max*. Este proceso de normalización se lleva a cabo ajustando los valores de las características al rango $[0, 1]$, utilizando los valores mínimos y máximos observados en el conjunto de datos de entrenamiento. Al analizar las características procesadas de *SATzilla*, se utilizó la matriz de correlación 22 para explorar las relaciones lineales entre las diferentes variables. En específico, se observó una alta correlación entre la cantidad de cláusulas y variables. Además, se identificó una significativa correlación entre los valores estadísticos de los nodos del grafo de variables, específicamente entre *vg_mean* (media de los valores) y *vg_max* (valor máximo)

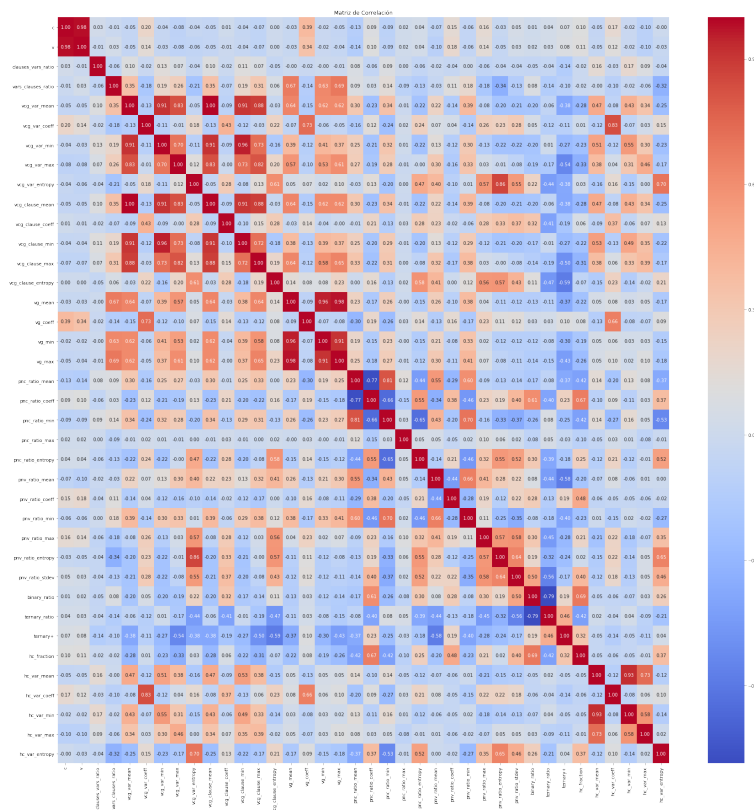


Figura 22: Matriz de correlación de las características de *SATzilla* Fuente: Elaboración propia

Por otra parte, en la figura 23 se presenta la información mutua entre las características y la clase a predecir. Destaca que la variable con el valor más alto de información mutua es la cantidad de variables. Este resultado señala que la cantidad de variables en una instancia tiene una fuerte relación informativa con la clase que se busca predecir.

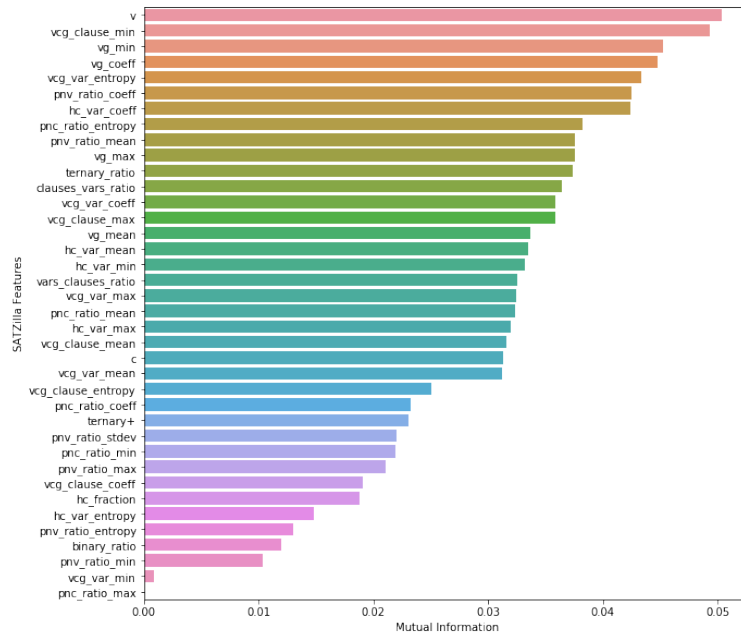


Figura 23: Gráfico de Información mutua entre las características de *SATZilla* y la variable objetivo Fuente: Elaboración propia

En cuanto a la caracterización con el historial de LBDs, se emplea la normalización detallada en el capítulo anterior. En la figura 24, se presentan las series de LBD de dos instancias distintas en su escala original. En contraste, la figura 25 muestra las mismas instancias con los datos normalizados. Este enfoque posibilita la observación de las series en escalas comparables, facilitando la identificación y comparación de sus comportamientos. La normalización resulta esencial para asegurar una representación equitativa de las instancias, permitiendo una interpretación más precisa y significativa del historial de LBDs en el contexto del modelo de *machine learning*.

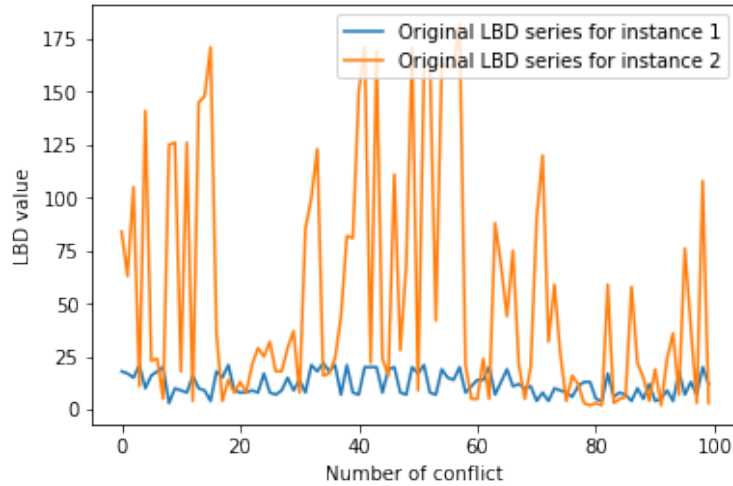


Figura 24: Gráfico de las curvas de valores de LBD originales para dos instancias distintas Fuente: Elaboración propia

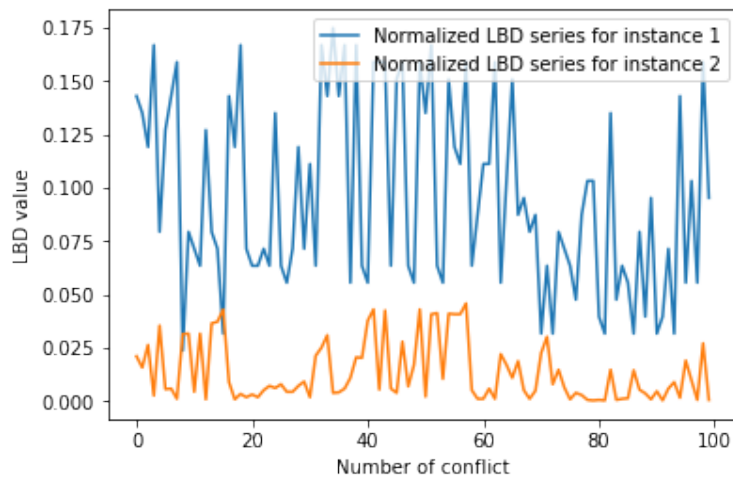


Figura 25: Gráfico de las curvas de valores de LBD normalizados a lo largo de la búsqueda para dos instancias distintas Fuente: Elaboración propia

5.3. Entrenamiento

Para el entrenamiento del modelo basado en *LSTM*, se llevaron a cabo 10 iteraciones, utilizando un tamaño de *batch* de 32. Se aprecia en la figura 26 una tendencia creciente en la métrica de *accuracy*, no obstante, se observa la presencia del fenómeno de sobreajuste. Este fenómeno sugiere que, a medida que el modelo sigue aprendiendo a partir de los datos de entrenamiento, su rendimiento mejora en ese conjunto particular, pero puede volverse menos generalizable a datos no vistos.

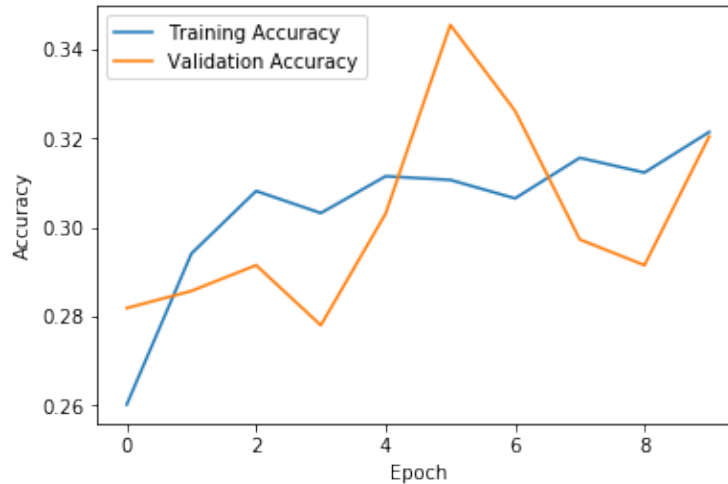


Figura 26: Gráfico del *accuracy* obtenido en cada iteración del entrenamiento Fuente: Elaboración propia

Para el modelo de clasificación *Random Forest*, se obtuvieron los valores de importancia de las características durante el entrenamiento, como se muestra en la figura 27. Contrario al análisis presentado en el procesamiento de datos, la cantidad de variables ya no figura como un factor relevante, llegando incluso a ubicarse entre las últimas. Entre las características más destacadas se encuentran los estadísticos de entropía, que emergen como factores significativos en la toma de decisiones del modelo. Este contraste en la importancia de las características entre el análisis de datos y el modelo entrenado resalta la capacidad del *Random Forest* para identificar patrones específicos y asignar relevancia de manera adaptativa durante la fase de entrenamiento.

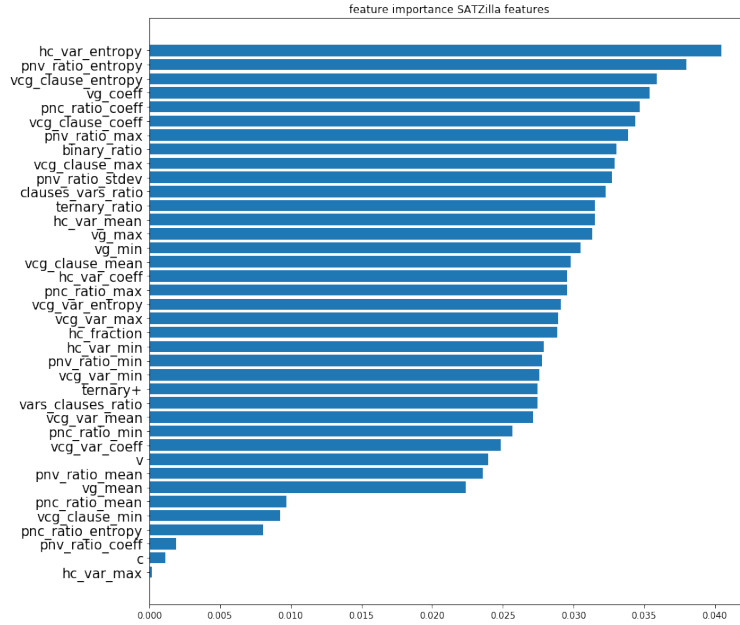


Figura 27: Gráfico de importancia de características del modelo *Random Forest* entrenado sobre características de *SATZilla* Fuente: Elaboración propia

5.4. Resultados

En esta sección se presentan los resultados al comparar los dos selectores de algoritmos entrenados en base a las caracterizaciones de *SATZilla* y el historial de LBD. La tabla 9 presenta las métricas de *accuracy*, *f1-score* y $\hat{m}$ para cada uno. El valor de $\hat{m}$ se midió en base al rendimiento de tiempo con *PAR10*, es decir, con penalización 10 veces el tiempo límite para instancias que no se pueden resolver. Se puede notar que tanto en el valor de *accuracy* como $\hat{m}$ los resultados fueron mejores en la caracterización en base al historial de LBD, aun así el valor de *f1 score* es mejor para el caso de *SATZilla*.

	SATZilla	LBD
accuracy	0.23	0.28
F1-Score	0.29	0.24
$\hat{m}$	0.92	0.81

Tabla 9: Métricas *accuracy*, *F1-Score* y $\hat{m}$ para los distintos selectores de algoritmos entrenados para la caracterización de *SATZilla* y LBD Fuente: Elaboración propia

Las figuras 28 y 29 presentan las matrices de confusión para los selectores basados en LBD y *SATZilla*, respectivamente. Ambas matrices muestran una tendencia a preferir un de las clase en la clasificación. En la primera matriz, las predicciones se concentran principalmente en la clase del SBS, mientras que una categoría no tiene asignada ninguna clasificación, lo que

impacta negativamente en las métricas de *accuracy* y *f1-score* del modelo. Por otro lado, en la segunda matriz, se observa que el modelo tiende a favorecer la primera clase, aunque los valores en la diagonal para las otras clases presentan una representación más precisa, explicando los mejores resultados en la métrica *f1 score*.

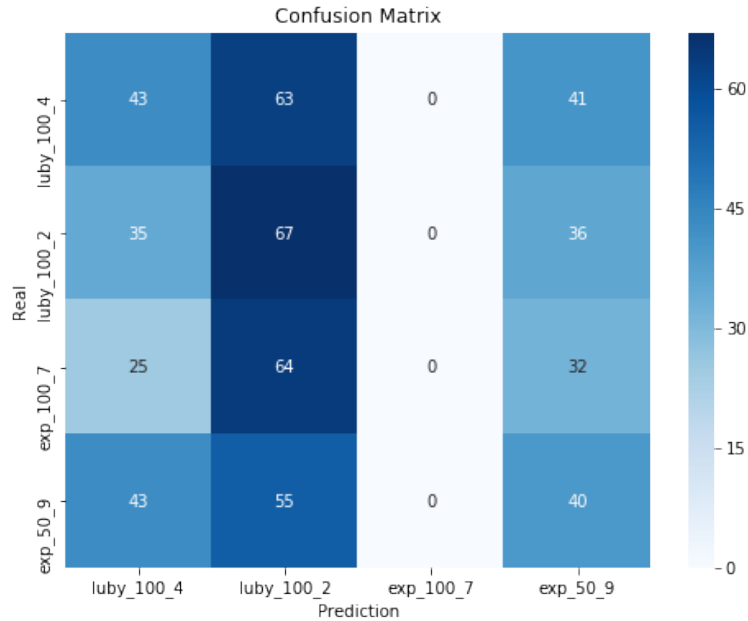


Figura 28: Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación para el modelo entrenado con el historial de LBD Fuente: Elaboración propia

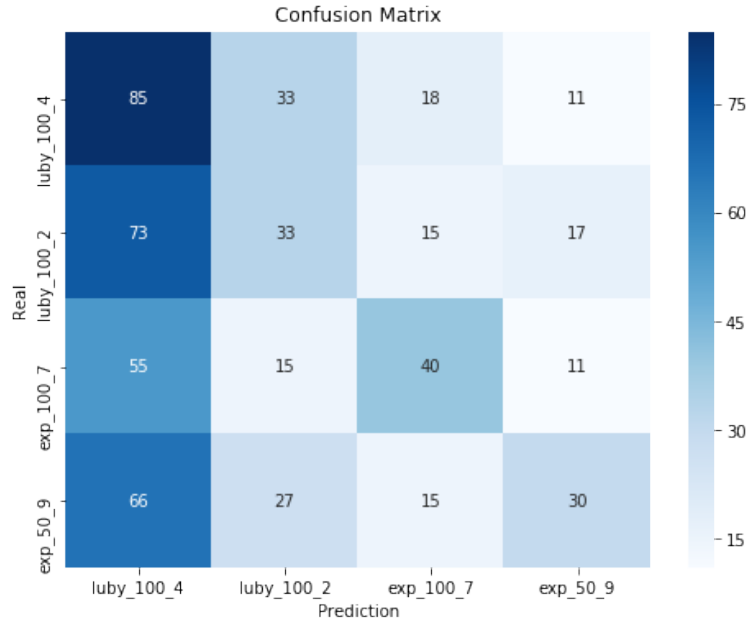


Figura 29: Histograma de las veces que un algoritmo gana sobre una instancia en el conjunto de validación para el modelo entrenado con las características de *SATZilla* Fuente: Elaboración propia

La Tabla 10 compara la cantidad de instancias resueltas por cada selector. Se observa que ambas caracterizaciones logran resolver más instancias que el SBS. No obstante, existe una marcada diferencia con el VBS. Esto también se ve reflejado la figura 30, si bien ambos selectores resuelven la misma cantidad de instancias, la caracterización a través de LBD muestra mejoras significativas en términos de tiempo de resolución.

	SATZilla	LBD	VBS	SBS
Cantidad de instancias resueltas	526	526	544	525

Tabla 10: Cantidad de instancias resueltas por los selectores de algoritmos entrenados para las caracterizaciones de *SATZilla* y LBD, comparados con su VBS y SBS sobre el conjunto de validacion Fuente: Elaboración propia

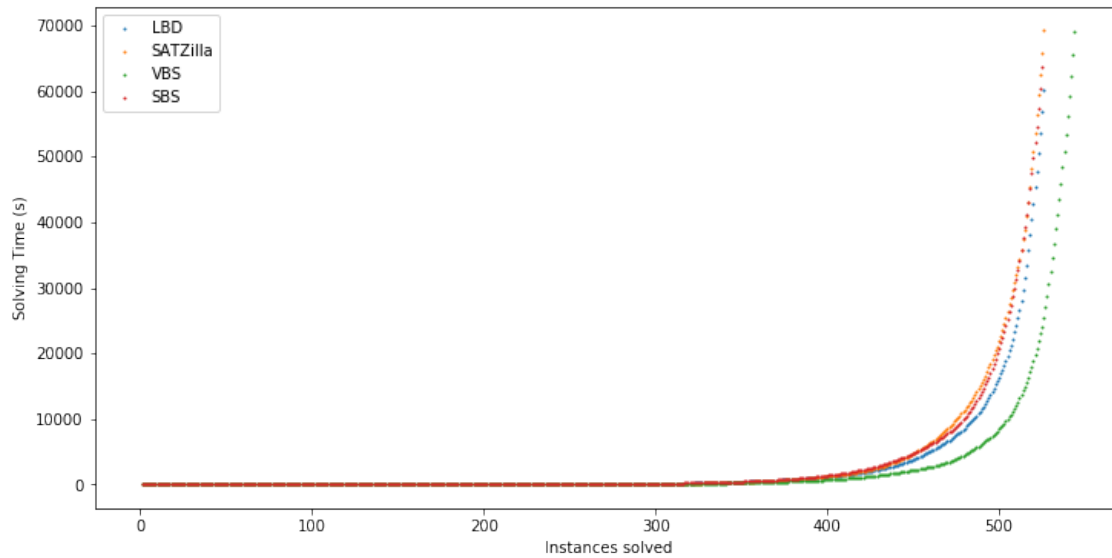


Figura 30: *Cactus-plot* de los selectores de algoritmos entrenados con LBD y SATZilla comparados con los VBS y SBS. Mientras más a la derecha está la curva, significa que resuelve más instancias, mientras más abajo está significa que resuelve las instancias más rápido Fuente: Elaboración propia

CAPÍTULO 6

CONCLUSIONES

6.1. Sobre los resultados

6.1.1. Construcción del *dataset*

Durante la construcción del *dataset*, se implementó con éxito la técnica de reinicio basada en el *solver* de *Glucose*, en el *solver* de *RoundingSAT*. Los experimentos realizados revelaron que esta técnica presenta un alto grado de complementariedad al comparar con la técnica basada en *Luby*. Aunque no resuelve un mayor número de instancias, demuestra su capacidad para acelerar la búsqueda en un conjunto significativo de instancias.

Asimismo, mediante la experimentación con las configuraciones de parámetros para las técnicas de *Luby* y *Glucose*, se evidenció que pequeñas variaciones en estos parámetros afectan el rendimiento del *solver*, tanto en la cantidad de instancias resueltas como en los tiempos de resolución.

En relación al *dataset* de instancias utilizado, el análisis indica que existe una considerable cantidad de instancias que son fáciles de resolver. Esto tuvo repercusiones en la obtención de caracterizaciones para las instancias, ya que, debido a su simplicidad, no se lograron registrar valores suficientes de LBD para todas las instancias.

Además, un resultado crucial en el análisis del *dataset* para el problema de clasificación de políticas de reinicio, es que se encontró que estas políticas no están correlacionadas con la pertenencia de la instancia a una familia específica. Esto implica que identificar la familia de una instancia no proporciona información suficiente para seleccionar la mejor política de reinicio para resolver esa instancia. Por lo tanto, este problema demuestra ser más robusto al evaluar los selectores de algoritmos.

6.1.2. Resultados del selector de algoritmos

En esta investigación se abordó la construcción y evaluación de selectores específicos para cada caracterización de instancias en el contexto de *SATZilla* y el historial de los valores de LBD. Para *SATZilla*, se implementó un *Random Forest*, mientras que para los LBD se utilizó una red neuronal basada en LSTM, con una propuesta de normalización para permitir la comparación de series de valores LBD con escalas diferentes.

Ambos selectores fueron evaluados mediante un conjunto de métricas que representan el aprendizaje de los modelos, así como métricas que permiten la comparación con los algorit-

mos del portafolio. Se observó una tendencia en ambas caracterizaciones hacia una de las clases; las características de *SATZilla* mostraron una distribución más favorable en la matriz de confusión, reflejándose en una mejor métrica $f1\text{-score}$. Este resultado puede estar influido por la limitada cantidad de instancias para entrenar modelos complejos como las LSTM, lo que podría conducir al sobreajuste del modelo. Además, el bajo impacto de una gran cantidad de instancias del *dataset* también podría haber contribuido.

Por otro lado, el modelo LSTM entrenado sobre el historial de LBD demostró mejoras en métricas como el *accuracy* y $\hat{m}$. Se destacó al superar al SBS tanto en tiempo de resolución como en la cantidad de instancias resueltas, posicionándose como una alternativa competitiva a la propuesta del estado del arte en caracterización de instancias para la selección de algoritmos.

En relación al rendimiento de *SATZilla* en la clasificación de políticas de reinicio, se observó un impacto negativo debido a la complejidad del problema, ya que clasificar la familia de la instancia no es suficiente para determinar la política de reinicio. Sin embargo, ambos modelos, a pesar de esta complejidad, demostraron aprender las políticas de reinicio para cada instancia, superando al SBS en términos de rendimiento.

En última instancia, la hipótesis que afirmaba que las LSTM entrenadas sobre el historial de LBD en *RoundingSAT* son capaces de seleccionar la técnica de reinicio en instancias PBO se respalda con los resultados obtenidos. Los modelos LSTM mostraron capacidad para aprender las técnicas de reinicio, y al compararlos con el SBS del portafolio y otro selector del estado del arte, se demostró que la propuesta del selector es competitiva y aporta mejoras significativas en la tarea de selección de técnicas de reinicio.

6.2. Contribuciones

Los resultados de la investigación han proporcionado varias contribuciones significativas en el ámbito de la caracterización y selección de instancias para problemas PBO.

La contribución principal radica en la propuesta de una nueva caracterización basada en el historial de LBD recopilados durante la búsqueda del solver *RoundingSAT*. Esta innovadora caracterización ha demostrado resultados prometedores al ser comparada con las características del estado del arte propuestas por *SATZilla*. Los resultados obtenidos sugieren que la nueva caracterización proporciona información valiosa y competitiva en la identificación de patrones en instancias PBO.

Además, el selector propuesto, entrenado utilizando la arquitectura de LSTM, ha demostrado su capacidad para aprender de las instancias. Como se evidencia en la sección de resultados, este selector presenta mejoras en términos de tiempos de resolución y cantidad de instancias resueltas en comparación con el SBS. Es destacable que también supera al selector construido en base al estado del arte propuesto en *SATZilla*, lo que subraya la importancia

de la política de reinicio en los *solvers* PBO.

Por otro lado, la implementación de la técnica de reinicio de *Glucose* en *RoundingSAT*, junto con una búsqueda de parámetros óptimos para ajustar el *solver*, resalta la vital importancia de la política de reinicio. Este avance pone en evidencia que incluso ajustes mínimos en los parámetros de esta política generan impactos significativos en la resolución de instancias. Enfatiza, de este modo, la importancia crítica de una elección precisa de la política de reinicio para alcanzar mejoras sustanciales en la eficacia del *solver*.

En conjunto, estas contribuciones avanzan significativamente en la comprensión y mejora de la resolución de instancias PBO, ofreciendo nuevas perspectivas y herramientas que tienen el potencial de impactar positivamente en el rendimiento de los *solvers* en situaciones prácticas y desafiantes.

6.3. Sobre los objetivos

En esta sección, retomaremos los objetivos establecidos al inicio de la investigación para llegar a conclusiones clave.

El primer objetivo, centrado en la implementación de técnicas de reinicio en el *solver RoundingSAT*, se logró con éxito mediante la introducción del reinicio basado en el *solver Glucose* para SAT. Los resultados evidencian mejoras sustanciales de esta técnica en un conjunto significativo de instancias, destacando una complementariedad significativa con la técnica por defecto basada en *Luby*.

El segundo objetivo, que buscaba analizar el rendimiento de técnicas de reinicio en el *solver RoundingSAT*, se abordó mediante experimentos que ajustaron configuraciones de los parámetros de las técnicas de reinicio *Luby* y *Glucose*. La sección de construcción del portafolio detalla los resultados en términos de cantidad de instancias resueltas, tiempo de resolución y complementariedad, culminando en la creación de un portafolio de algoritmos.

El tercer objetivo, enfocado en proponer una caracterización de instancias PBO, se cumplió con la introducción de una nueva caracterización que considera el comportamiento del historial de valores de LBD durante la ejecución del algoritmo basado en CDCL en *RoundingSAT*. Los resultados presentados en la sección correspondiente demuestran que estas características son competitivas al compararlas con las definidas en el estado del arte propuesto en *SATZilla*.

El cuarto objetivo, orientado a implementar un selector de algoritmo para elegir la técnica de reinicio del *solver RoundingSAT* para una instancia PBO dada, se logró mediante el diseño de un selector basado en redes neuronales recurrentes LSTM. Este selector se entrenó con la caracterización propuesta basada en LBD, con el objetivo de predecir la mejor técnica de reinicio.

Finalmente, el último objetivo, que tiene como propósito evaluar el desempeño del selector comparándolo con el estado del arte, se abordó construyendo un segundo selector basado en el estado del arte. Este nuevo selector, basado en un modelo *Random Forest* y entrenado con la caracterización de *SATZilla*, se comparó con el selector propuesto. Los resultados presentados en la sección correspondiente muestran mejoras significativas en comparación con el SBS y el selector basado en *SATZilla*, validando así la eficacia del enfoque propuesto. En conjunto, estos logros cumplen los objetivos establecidos, subrayando el avance significativo logrado en el ámbito de la selección de técnicas de reinicio para solvers PBO.

6.4. Limitaciones

A pesar de los valiosos hallazgos y contribuciones, es crucial abordar y reconocer las limitaciones inherentes que podrían influir en la interpretación de los resultados. Una limitación fundamental, analizada a lo largo de la investigación, se relaciona con la calidad de las instancias recopiladas. La mayoría de estas instancias mostró una baja complejidad, limitando así el entrenamiento de los modelos a unas pocas instancias que realmente impactaron en los tiempos de resolución y en las clases seleccionadas por los selectores.

Adicionalmente, es importante señalar que no fue posible caracterizar completamente todas las instancias, ya que una parte de ellas finalizaban la búsqueda antes de generar los LBD necesarios para entrenar el modelo. Esta limitación plantea desafíos en la representación integral de todas las clases y comportamientos posibles, afectando la generalización de los modelos.

Asimismo, es esencial considerar la complejidad asociada con la transformación de instancias PBO a CNF. En este estudio, se utilizaron únicamente las características básicas de *SATZilla* debido a la complejidad de este proceso. Esta limitación implica que la propuesta de caracterización no se compara exhaustivamente con la propuesta de *SATZilla*, lo que podría limitar la comprensión completa de las características distintivas de las instancias PBO.

En conjunto, estas limitaciones destacan áreas clave para futuras investigaciones y resaltan la importancia de abordar la complejidad de las instancias PBO para lograr una caracterización más completa y precisa.

6.5. Trabajo Futuro

En conclusión, se ha proporcionado valiosas perspectivas sobre diversos aspectos del proceso de selección de algoritmos para el problema PBO. No obstante, persisten áreas sin explorar y preguntas que abren oportunidades para futuras investigaciones.

Por un lado, se destaca la importancia de las técnicas de reinicio, aunque este portafolio solo

abordó dos técnicas, existen otras alternativas que merecen una evaluación exhaustiva. Además, la definición de un subconjunto pequeño de parámetros sugiere que una exploración más detallada de estos podría conducir a resultados más robustos y eficaces.

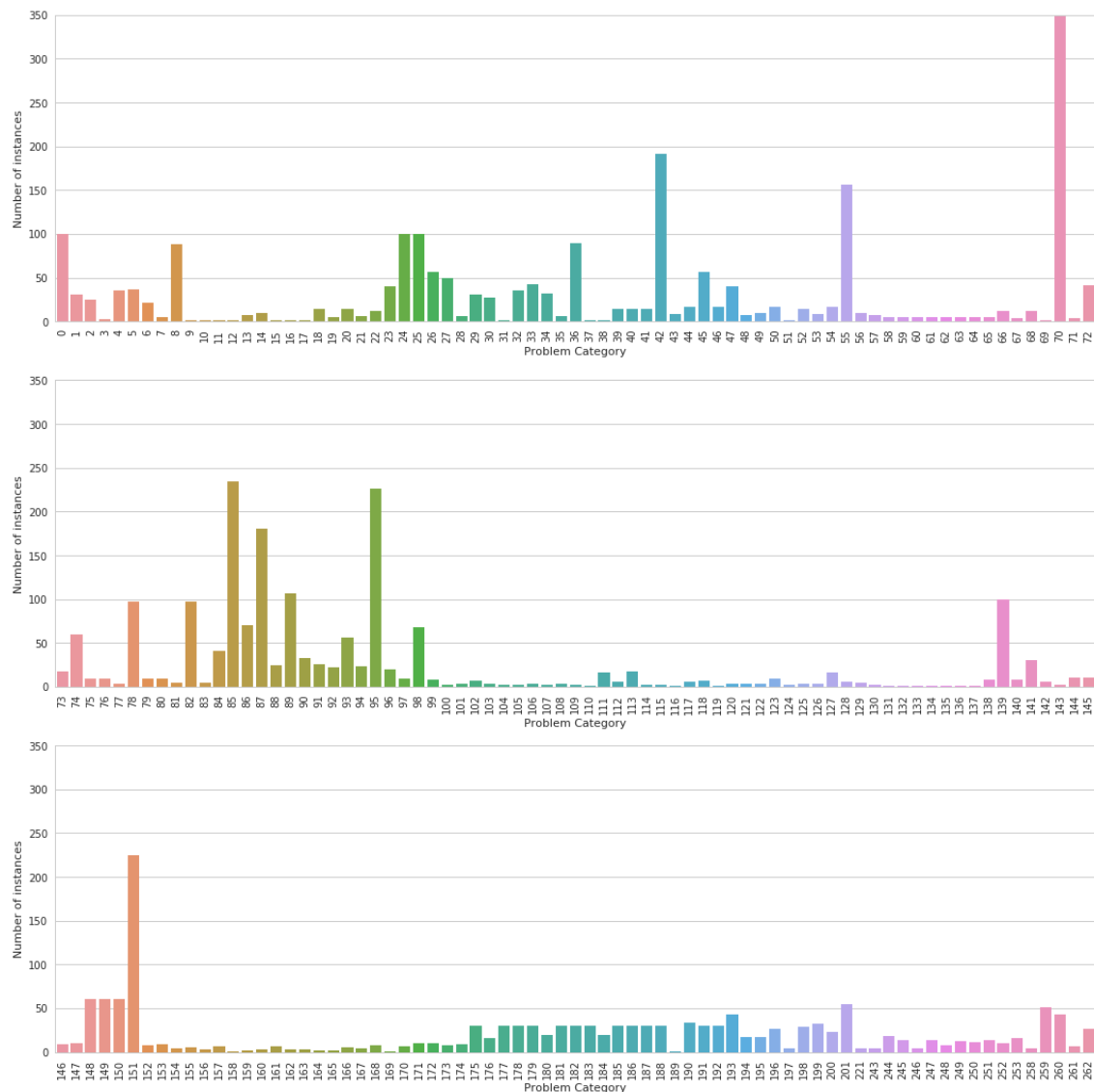
Por otro lado, es esencial evaluar otras técnicas de procesamiento de las series de valores LBD que mejoren la identificación de patrones en estas secuencias numéricas. La exploración de diferentes modelos de *machine learning* para identificar patrones podría abrir nuevas perspectivas en este sentido. Se sugiere probar con las características completas de *SATZilla*, así como explorar otras caracterizaciones actuales que han demostrado ser competitivas.

Asimismo, se recomienda llevar a cabo estudios adicionales utilizando otros conjuntos de datos. Por ejemplo, la obtención de datos más complejos y en mayor cantidad para instancias del problema SAT podría contribuir significativamente a mejorar los modelos entrenados. Este enfoque permitiría una comprensión más completa de los desafíos y patrones presentes en diferentes escenarios, fortaleciendo así la generalización de los modelos.

En resumen, este estudio sienta las bases para futuras investigaciones que podrían abordar de manera más exhaustiva las diversas facetas del proceso de selección de algoritmos para el problema PBO, brindando así un mayor entendimiento y mejores estrategias para mejorar la eficiencia de los *solvers* en este contexto.

ANEXOS

Anexo 1: Diagrama de barras con la cantidad de instancias por familia



REFERENCIAS BIBLIOGRÁFICAS

- [Alfonso y Manthey, 2018] Alfonso, E. M. y Manthey, N. (2018). New CNF features and formula classification. *EPiC series in computing*.
- [Ansótegui *et al.*, 2017] Ansótegui, C., Bonet, M. L., Giráldez-Cru, J., y Levy, J. (2017). Structure features for SAT instances classification. *Journal of Applied Logic*, 23:27–39.
- [Applegate *et al.*, 2007] Applegate, D., Bixby, R. E., Chvátal, V., y Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*.
- [Arora y Barak, 2009] Arora, S. y Barak, B. (2009). *Computational Complexity*. Cambridge University Press.
- [Audemard *et al.*, 2011] Audemard, G., Lagniez, J.-M., Mazure, B., y Saïs, L. (2011). On freezing and reactivating learnt clauses. *HAL (Le Centre pour la Communication Scientifique Directe)*.
- [Audemard y Simon, 2009] Audemard, G. y Simon, L. (2009). Predicting learnt clauses quality in modern sat solvers. pp. 399–404.
- [Audemard y Simon, 2012] Audemard, G. y Simon, L. (2012). Refining restarts strategies for sat and unsat. En Milano, M., editor, *Principles and Practice of Constraint Programming*, pp. 118–126, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Berre y Wallon, 2021] Berre, D. L. y Wallon, R. (2021). On dedicated CDCL strategies for PB solvers. *arXiv (Cornell University)*.
- [Biere y Fröhlich, 2015] Biere, A. y Fröhlich, A. (2015). *Evaluating CDCL variable scoring schemes*.
- [Bordeaux *et al.*, 2006] Bordeaux, L., Hamadi, Y., y Lintao, Z. (2006). Propositional satisfiability and constraint programming: A comparative survey. *ACM Comput. Surv.*, 38.
- [Boros y Hammer, 2002] Boros, E. y Hammer, P. L. (2002). Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225.
- [Breiman, 1984] Breiman, L. (1984). Classification and regression trees. *Biometrics*, 40(3):874.
- [Breiman, 2001] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.
- [Chen y Guestrin, 2016] Chen, T. y Guestrin, C. (2016). Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM.

- [Devriendt, 2020] Devriendt, J. (2020). Watched propagation of 0-1 integer linear constraints. En Simonis, H., editor, *Principles and Practice of Constraint Programming*, pp. 160–176, Cham. Springer International Publishing.
- [Devriendt et al., 2021a] Devriendt, J., Gleixner, A. M., y Nordström, J. (2021a). Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints - An International Journal*, 26(1-4):26–55.
- [Devriendt et al., 2021b] Devriendt, J., Gocht, S., Demirović, E., Nordström, J., y Stuckey, P. J. (2021b). Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3750–3758.
- [Diestel, 2017] Diestel, R. (2017). *Graph theory*.
- [El Naqa y Murphy, 2015] El Naqa, I. y Murphy, M. J. (2015). *What Is Machine Learning?*, pp. 3–11. Springer International Publishing, Cham.
- [Elffers y Nordström, 2018] Elffers, J. y Nordström, J. (2018). Divide and conquer: Towards faster pseudo-boolean solving. En *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 1291–1299. International Joint Conferences on Artificial Intelligence Organization.
- [Eén y Sörensson, 2006] Eén, N. y Sörensson, N. (2006). Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26.
- [Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [Haim y Walsh, 2009] Haim, S. y Walsh, T. (2009). *Restart strategy selection using machine learning techniques*.
- [Hastie et al., 2009] Hastie, T., Tibshirani, R., y Friedman, J. H. (2009). *The elements of statistical learning*.
- [Hochreiter y Schmidhuber, 1997] Hochreiter, S. y Schmidhuber, J. (1997). Long Short-Term memory. *Neural Computation*, 9(8):1735–1780.
- [Hoos, 2005] Hoos, H. H. (2005). *PROPOSITIONAL SATISFIABILITY AND CONSTRAINT SATISFACTION*.
- [Huang, 2007] Huang, J. (2007). The effect of restarts on the efficiency of clause learning. pp. 2318–2323.
- [Jia et al., 2018] Jia, L., Oh, C., Mathew, M., Thomas, C., Li, C., y Ganesh, V. (2018). *Machine Learning-Based Restart Policy for CDCL SAT solvers*.

- [Kerschke et al., 2019] Kerschke, P., Hoos, H. H., Neumann, F., y Trautmann, H. (2019). Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45.
- [Kingma y Ba, 2014] Kingma, D. P. y Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv (Cornell University)*.
- [Lawler y Wood, 1966] Lawler, E. E. y Wood, D. (1966). Branch-and-Bound Methods: A survey. *Operations Research*, 14(4):699–719.
- [Liang et al., 2018] Liang, J., Oh, C., Mathew, M., Thomas, C., Li, C., y Ganesh, V. (2018). *Machine Learning-Based Restart Policy for CDCL SAT Solvers*. Springer Science+Business Media.
- [Liu et al., 2020] Liu, M.-H., Zhang, F., Huang, P. Y., Niu, S., Ma, F., y Zhang, J. (2020). *Learning the Satisfiability of Pseudo-Boolean Problem with Graph Neural Networks*. Springer Science+Business Media.
- [Martello y Toth, 1991] Martello, S. y Toth, P. (1991). Knapsack problems: Algorithms and computer implementations. *Journal of the Operational Research Society*, 42(6):513.
- [Murphy, 2022] Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.
- [Nemhauser y Wolsey, 1988] Nemhauser, G. L. y Wolsey, L. A. (1988). *Integer and combinatorial optimization*.
- [Nordström, 2021] Nordström, J. (2021). Pseudo-Boolean Solving and Optimization.
- [Nudelman et al., 2004] Nudelman, E., LeytonBrown, K., Hoos, H. H., Devkar, A., y Shoham, Y. (2004). *Understanding random SAT: Beyond the Clauses-to-Variables ratio*.
- [Patro y Sahu, 2015] Patro, S. G. K. y Sahu, K. K. (2015). Normalization: a preprocessing stage. *International advanced research journal in science, engineering and technology*, pp. 20–22.
- [Pipatsrisawat y Darwiche, 2009] Pipatsrisawat, K. y Darwiche, A. (2009). *Width-Based Restart Policies for Clause-Learning Satisfiability Solvers*.
- [Provan-Bessell et al., 2022] Provan-Bessell, B., Dalla, M., Visentin, A., y O'Sullivan, B. (2022). SATfeatPY – a Python-based feature extraction system for satisfiability. *arXiv (Cornell University)*.
- [Sakai y Nabeshima, 2015] Sakai, M. y Nabeshima, H. (2015). Construction of an ROBDD for a PB-Constraint in band form and related techniques for PB-Solvers. *IEICE Transactions on Information and Systems*, E98.D(6):1121–1127.
- [Woeginger, 2003] Woeginger, G. J. (2003). *Exact Algorithms for NP-Hard Problems: A Survey*.

- [Xu *et al.*, 2008] Xu, L., Hutter, F., Hoos, H. H., y LeytonBrown, K. (2008). SATZilla: Portfolio-based Algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.
- [Zhang *et al.*, 2001] Zhang, L., Madigan, C., Moskewicz, M., y Malik, S. (2001). Efficient conflict driven learning in a boolean satisfiability solver. En *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pp. 279–285.