UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
Programa de Magíster en Ciencias de la Computación

SELECCIÓN AUTOMÁTICA DE ALGORITMOS ANYTIME PARA COLOREO DE GRAFOS

Tesis presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al grado de Magíster en Ciencias de la Computación

POR: DANIEL ANTONIO ORTEGA CÁRCAMO
PROFESOR GUÍA : ROBERTO JAVIER ASÍN ACHÁ

Enero, 2021
Concepción, Chile

Índice General

Índice de Tablas

Índice de Ilustraciones

RESUMEN

Las distintas técnicas de solución a problemas de optimización combinatoria permiten abordar problemas computacionalmente difíciles. Sin embargo, de manera general, la pertinencia de cada técnica depende de cada escenario específico. Este escenario puede estar compuesto por una instancia particular del problema y de recursos computacionales concretos. Dependiendo de la combinación de estos factores, no es trivial identificar cuál de las técnicas de solución es la más adecuada.

Debido a lo anterior, en esta tesis se propone desarrollar un "oráculo", basado en técnicas de machine learning, capaz de discriminar, dentro de un portafolio de alternativas, cuál es la más adecuada para solucionar una instancia específica considerando un tiempo límite de ejecución concreto. El dominio de aplicación de este oráculo es el Problema de Coloreo de Grafos, que es un problema ampliamente estudiado y que presenta interés tanto a nivel teórico como práctico.

En la presente investigación se presenta el trabajo desarrollado en relación a la generación de datos de entrenamiento y validación, la caracterización de instancias y el proceso completo de aprendizaje de máquina asociado a estos datos. Se propusieron, desarrollaron y evaluaron diversos modelos de aprendizaje, basados tanto en clasificación como en regresión. Estos modelos fueron optimizados de acuerdo a diferentes métricas y se analizaron sus resultados para discriminar la alternativa más adecuada a la tarea. Finalmente, el modelo más apropiado fue evaluado en un escenario tipo competencia y se pudieron extraer conclusiones respecto a la efectividad y utilidad práctica del enfoque propuesto.

Entre los modelos generados, el más adecuado corresponde a un modelo de clasificación, guiado por la métrica GM. Para éste se identificó un conjunto significativo (mínimo) de características de entrada, con el objetivo de disminuir lo más posible el tiempo necesario de predicción, sin perder calidad en la recomendación del método de solución más adecuado.

Los resultados nos permiten concluir que es posible desarrollar modelos de aprendizaje efectivos para la tarea. Sin embargo, en el caso específico de coloreo de grafos, su utilidad práctica es limitada, debido al nivel de dominancia que tienen los solvers *Head* y *HybridEA* a lo largo del conjunto de datos. Otro desafío importante es minimizar el tiempo necesario para extraer las características de entrada de los modelos de aprendizaje y hacer el cómputo necesario para la predicción del oráculo.

ABSTRACT

The different techniques for solving combinatorial optimization problems allow us to tackle computationally difficult problems. However, in general, the relevance of each technique depends on each specific scenario. This scenario may be composed of a particular instance of the problem and of specific computational resources. Depending on the combination of these factors, it is not trivial to identify which of the solution techniques is the most appropriate.

Due to the above, this thesis proposes to develop an "oracle", based on machine learning techniques, capable of discriminating, within a portfolio of alternatives, which is the most adequate to solve a specific instance considering a concrete execution time limit. The application domain of this oracle is the Problem of Graph Coloring, which is a widely studied problem that presents interest both at a theoretical and practical level.

In the present research, the work developed in relation to the generation of training and validation data, the characterization of instances and the complete machine learning process associated with these data is presented. Several learning models were proposed, developed and evaluated, based on both classification and regression. These models were optimized according to different metrics and their results were analyzed to discriminate the most appropriate alternative to the task. Finally, the most appropriate model was evaluated in a competition-type scenario and conclusions could be drawn regarding the effectiveness and practical utility of the proposed approach.

Among the models generated, the most appropriate corresponds to a classification model, guided by GM metrics. For this one, a significant set of (minimum) input characteristics was identified, with the objective of reducing the necessary prediction time as much as possible, without losing quality in the recommendation of the most appropriate solution method.

The results allow us to conclude that it is possible to develop effective learning models for the task. However, in the specific case of graph coloring, its practical utility is limited, due to the level of dominance that have the solvers *Head* and *HybridEA* along the data set. Another important challenge is to minimize the time needed to extract the input features from the learning models and to do the computation necessary for oracle prediction.

1. INTRODUCCIÓN

1.1. Contexto

Muchos problemas de la vida real pueden ser modelados como problemas de Optimización Combinatoria. Una gran variedad de estos problemas pertenece a la clase de problemas NP-hard, lo que significa que, muy probablemente, sean intratables computacionalmente. A pesar de ello, existen muchas propuestas algorítmicas, exactas (exponenciales en el peor de los casos) y aproximadas, para muchos de estos problemas. En general, experimentalmente, en este tipo de problemas, el rendimiento comparativo de los algoritmos depende fuertemente de la instancia específica a ser resuelta, del tiempo límite que estemos dispuestos a esperar por la respuesta e, incluso, de los recursos de hardware con los que contemos. Así, es posible que para una instancia específica de cierto problema y utilizando el mismo computador, un algoritmo (solver) arroje la mejor solución con 10 segundos de cómputo y sea otro el que encuentre la mejor solución después de ese tiempo.

Lo anterior, se conoce como teoremas "No Free Launch (NFL)" [1, 2], que se resume en "Dado un problema general, no existe una única técnica que supere a las demás en todos los escenarios posibles". Estos escenarios pueden considerar variaciones en el tipo de instancias a ser resueltas y los recursos computacionales disponibles (e.g. tiempo de ejecución, memoria disponible, frecuencia del procesador, arquitectura, etc). La existencia de este tipo de observaciones plantea a la comunidad que trabaja en este tipo de problemas, un nuevo problema: escoger el mejor método de solución específico para cada escenario. El poder resolver este nuevo problema permitiría, en la práctica, lidiar de mejor forma con cada escenario y otorgar, a los poseedores de los problemas de optimización, mejores soluciones para ser implementadas en la práctica y ahorrar recursos computacionales.

Para lograr un solver que entregue la mejor solución posible ante un escenario dado, se han estudiado técnicas como el uso de portafolio de algoritmos (ejecutar todos los solvers al mismo tiempo) y la selección automática de algoritmos [3]. Ésta última pretende usar un "oráculo" basado en técnicas de machine learning, para decidir qué solver obtendrá el mejor rendimiento para un escenario específico y usarlo como técnica de solución. Ambos enfoques han demostrados resultados prometedores, sin embargo, para esta tesis se optará por una

modificación del segundo.

Entre la gran diversidad de escenarios de cómputo a considerar, aparte de los que varían las características específicas de cada instancia, uno muy relevante es variar el tiempo para encontrar una solución a una instancia específica. En la práctica, existen aplicaciones que requieren soluciones suficientemente buenas en un tiempo muy acotado, mientras que otras pueden permitirse el uso de horas e incluso días, con el objetivo de encontrar la solución óptima. La forma en la que evoluciona en el tiempo la calidad de la mejor solución encontrada por un solver se conoce como el comportamiento anytime del mismo [4]. En este trabajo, a diferencia de trabajos similares en la literatura, usamos esta importante característica como parámetro de decisión para realizar la recomendación del mejor solver posible para cada situación.

Este trabajo estudia el problema de selección automática de algoritmos, en función de su comportamiento anytime, para una gran variedad de instancias del problema de Coloreo de grafos o Graph Coloring Problem (GCP). GCP pertenece a la lista de problemas NP-completos de Karp [5] y consiste en encontrar un número limitado de colores para colorear un grafo de tal manera que los colores no se repitan entre nodos vecinos. Este problema puede ser aplicado en diversas áreas, como: colas de programación [6, 7, 8], horarios [9, 10], asignación de registros [11] y redes de comunicación [12], entre otros.

En esta tesis se propone utilizar distintas instancias para el problema de coloreo de grafos, registrando el comportamiento anytime de distintos solvers que pertenezcan a diferentes familias (paradigmas) de solución. El comportamiento anytime de cada solver, en cada instancia de prueba, fue registrado para construir el conjunto de datos sobre los que se entrenaron diversos modelos de machine learning. Los modelos debían aprender a seleccionar el algoritmo más adecuado para resolver una instancia específica, considerando una restricción temporal. Para diferenciar cada instancia, se calcularon distintas características, que permitan discriminar el solver más adecuado para la resolución de la instancia, con el mejor resultado posible.

A lo largo de esta tesis se presentan distintos modelos de clasificación y regresión para el problema planteado. Se propusieron distintos modelos que fueron evaluados bajo diferentes métricas. Un foco importante de los modelos de clasificación fue lidiar con el desbalance de las clases y equilibrar el error a lo largo de los distintos solvers candidatos. De igual forma, para la evaluación de estos modelos, a diferencia de lo usualmente reportado en la literatura, se consideró el tiempo necesario para el cálculo de las características de las instancias y el tiempo

de ejecución del modelo. Esto último es importante, con el fin de evaluar la utilidad práctica de este tipo de enfoques en la situación actual del estado del arte en Coloreo de Grafos.

Los resultados obtenidos nos permiten afirmar que es posible diseñar y entrenar modelos de machine learning que permitan discriminar entre qué método de solución es el más adecuado ante distintos escenarios que varían el tipo de instancia y el tiempo de ejecución disponible para los métodos de solución. A pesar del desbalance existente en el conjunto de datos, se pudieron entrenar modelos de clasificación altamente efectivos y rápidos, capaces de discriminar las clases más desfavorecidas. Sin embargo, debido a que el estado del arte en la resolución de problemas de coloreo de grafos presenta un método de solución ampliamente dominante, lo presentado en esta tesis puede ser considerado de uso general y aplicable a otros dominios e incluso, llegar a ser de uso práctico cuando existan nuevos solvers que permitan lograr un mejor balance en los datos, haciendo que el enfoque presentado en este trabajo sea de utilidad.

1.2. Hipótesis

Es posible crear un modelo que seleccione adecuadamente el mejor solver para una instancia dada de coloreo de grafos considerando un tiempo límite de ejecución variable.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un modelo de machine learning que seleccione el método que entregue la mejor solución posible para una instancia específica del problema de coloreo de grafos, dado un tiempo límite de ejecución.

1.3.2. Objetivos específicos

- Generar un conjunto de datos a partir del registro del comportamiento anytime de solvers del estado del arte, sobre instancias de distinta naturaleza de coloreo de grafos.
- Diseñar e implementar modelos de machine learning que aprenda de los datos generados.

- Evaluar los resultados sobre métricas estándar de machine learning.
- Evaluar los resultados bajo escenarios tipo competencia, en los que se considere el tiempo de cálculo de características y de predicción del modelo.
- Comunicar los resultados obtenidos durante la investigación.

1.4. Limitaciones

Se consideraron las siguientes limitaciones para la realización de este trabajo.

Gran parte de los resultados reportados en el estado del arte son evaluados en distintas arquitecturas de computadores con distintos criterios de término de la ejecución del algoritmo como: tiempo límite, cantidad máxima de iteraciones o cantidad de restricciones revisadas. Esto complica determinar que trabajo es más útil que otro. Por ello, se realizó una revisión exhaustiva del estado del arte y se consideraron solvers listados en trabajos similares, más algunos otros publicados en los últimos años.

Muchos de los solvers propuestos en el estado del arte no tienen su código fuente disponibles. Pese a que se solícito el código fuente a los autores, no se obtuvo respuesta. Hay que considerar que el código fuente es necesario para modificar los solvers con tal de registrar su comportamiento anytime.

La implementación de los solvers utilizada en este trabajo, es la implementación realizada por los autores usando el lenguaje y optimización que ellos propusieron. Queda fuera del alcance de este trabajo realizar comparaciones sobre que lenguaje de programación o que optimizador es más eficiente.

Los parámetros con los cuales se ejecutaron los solvers son los reportados por defecto por los mismos autores.

Las técnicas de entrenamiento de los modelos de machine learning fueron limitadas a aquellas capaces de dar una explicación sobre el conjunto de características sobre las que se realiza la predicción. Esto, debido a que se considera importante estudiar y comprender cómo las características (el tiempo de ejecución incluido entre ellas) afectan la predicción de qué solver tiene un mejor comportamiento.

1.5. Estructura del informe

Esta investigación se estructura en el siguiente orden:

- Capítulo 1. Introducción: Se entrega la motivación de la investigación, se expone el problema a tratar y cómo abordarlo.
- Capítulo 2. Marco Teórico: Se describen conceptos teóricos relevantes para el desarrollo de la investigación.
- Capítulo 3. Revisión del Estado del Arte: Se describe lo encontrado en la revisión literaria de selección automática de algoritmos sobre coloreo de grafos.
- Capítulo 4. Solución Propuesta y resultados: Se describe el marco de trabajo propuesto, junto con los datos obtenidos, instancias y solvers a utilizar.
- Capitulo 5. Resultados: Se presentan los resultados obtenidos en los modelos al simular competencias evaluando distintos tiempos limites.
- Capítulo 6. Conclusiones y Trabajo Futuro: Se describen los resultados de la investigación y posibles líneas para trabajos futuros.

2. MARCO TEÓRICO

2.1. Coloreo de grafos

2.1.1. Definición y condiciones

Sea un grafo no dirigido y acíclico $G = (V, E)$, con un conjunto de vértices o nodos V y un conjunto de aristas E, siendo una arista $(u, v) : u, v \in V$. Dado un grafo, este problema busca asignar a cada vértice $v \in V$, un número $\{1, 2, \ldots, k\}$, con k la cantidad máxima de colores a utilizar, mediante la función $c(v)$, tal que:

1. $c(u) \neq c(v)$, $\forall \{u, v\} \in E$.
2. k es mínimo.

Un coloreo puede ser considerado *legal* o *factible* si la restricción uno se cumple; por el contrario un coloreo puede ser considerado *infactible*, en el caso de que dos nodos adyacentes tengan el mismo color (es decir $c(u) = c(v)$). El mínimo número de colores con que todos los vértices pueden ser coloreados respetando las restricciones anteriores es conocido como el número cromático de G, denotado por $\chi(G)$. En resumen, el problema de coloreo de grafos corresponde a determinar el valor de $\chi(G)$ y encontrar una coloración factible de G que use $\chi(G)$ colores. Además, se puede entender un coloreo como una partición de V en k subconjuntos, donde todos los nodos en un mismo subconjunto $S_x, x \in \chi(G)$ tiene el mismo color x. Entonces, un coloreo es nombrado *legal* si para cada subconjunto S_x no hay nodos adyacentes entre sí. Estos subconjuntos son llamados *conjuntos independientes*.

Para dimensionar la complejidad de este problema, es factible enumerar las soluciones posibles: primero siguiendo un criterio de asignación se puede asignar un color distinto a cada nodo, lo cual es ineficiente considerando que existen n^n, con $n = |V|$, soluciones posibles a ser verificadas, sin considerar las simetrías que se producen. Segundo, una mejor formulación del problema es la de Maximal Independient Set (MIS), con él se evitan simetrías, si se limita el número de colores k a utilizar, la cantidad de soluciones queda definida por el *número de Stirling* $\left\{ \begin{array}{c} n \\ k \end{array} \right\}$, que define el número de formas en las que se puede particionar n elementos

en k subconjuntos no vacíos. Si se considera un algoritmo que intente enumerar desde 1 hasta $\chi(G)$, se tendría que revisar a lo más $\sum_{k=1}^{\chi(G)} \left\{ \begin{array}{c} n \\ k \end{array} \right\}$ soluciones posibles [13].

2.1.2. Cotas

Se puede ver que verificar todas las soluciones puede requerir un tiempo considerable; por ende, para tener un mejor entendimiento del problema se buscan las *cotas*. En el caso de una *cota inferior*, se debe observar que si un grafo contiene un subgrafo completo, (cada par de vértices en V esta conectado por una arista denotado por K_k [14]), entonces un coloreo factible requerirá como mínimo k colores, de la misma manera que el número de nodos contenidos en la clique [14] más larga de G, denotado por $\omega(G)$; lo cual corresponde a una *cota inferior*: $\chi(G) \geq \omega(G)$.

Si se analiza desde la formulación de conjuntos y se considera el número de vértices contenido en el conjunto independiente más grande en G, conocido como el *número de independencia* y denotado por $\alpha(G)$, se tiene $\chi(G) \geq \lceil n/\alpha(G) \rceil$. Finalmente juntando $\alpha(G)$ y $\omega(G)$, se tiene que una *cota inferior* que esta dada por: $\chi(G) \geq \text{máx}\{\omega(G), \lceil n/\alpha(G) \rceil\}$ [13].

Para definir una *cota superior*, se debe observar los grados de los vértices en un grafo, ya que, si un grafo tiene una mayor densidad (existe una gran cantidad de aristas entre nodos), se requerirá una mayor cantidad de colores para etiquetar los vértices con distintos colores. Lo anterior puede ser resumido en un teorema de teoría de grafos: *Sea G un grafo conexo con un grado máximo de* $\Delta(G) = \text{máx}\{\deg(v) : v \in V\}$*, entonces* $\chi(G) \leq \Delta(G) + 1$ [13]. De esta manera el problema a tratar queda acotado por:

$$\text{máx}\{\omega(G), \lceil n/\alpha(G) \rceil\} \leq \chi(G) \leq \Delta(G) + 1 \tag{1}$$

2.1.3. Algoritmos en la literatura

Tal como se hizo notar en lo párrafos anteriores el GCP es un problema NP-Completo [15] y tiene relación directa con otros problemas NP-Hard como: la maximum clique [16] y el MIS. Además se han propuesto distintas variaciones mencionadas en [17]. Es importante destacar

que a diferencia de otros problemas NP-hard, este problema es difícil en promedio, lo que quiere decir que las instancias aleatorias tienden a ser difíciles de resolver [18].

A pesar de la dificultad del problema, se han desarrollado distintos enfoques con ventajas y desventajas que permitirían resolverlo de forma aproximada.

Hay que considerar que algunas instancias con ciertas estructuras pueden ser resueltas en tiempo polinomial, pero esto no sucede en la mayoría de los casos. Por ende, para abordar de mejor manera el problema, se tienen dos enfoques de algoritmos en la literatura: *los exactos y los aproximados*.

Los *exactos*, suelen ser aplicados en grafos pequeños. Estos algoritmos suelen basarse en búsquedas exhaustivas que acotan el espacio de búsqueda saltándose áreas no prometedoras o reduciendo el número de recálculos [19]. En está línea se ha estudiado el uso de algoritmos como *branch and bound*, *programación dinámica* y algoritmos de *programación lineal entera* [20]. Entre ellos destacan técnicas como:

- **Backtracking:** Desde su proposición formal en [21], esta técnica se ha utilizado principalmente para resolver el Constraint Satisfaction Problem (CSP). Se basa en transformar sistemáticamente soluciones parciales en soluciones completas. Durante el proceso de construcción si existe evidencia de que no hay forma de completar la solución parcial, el algoritmo realiza un retroceso (backtrack) para encontrar formas adecuadas de ajustar la actual solución parcial .
- **Programación Lineal Entera:** Permite resolver problemas donde se debe encontrar valores enteros a variables, maximizando o minimizando una función lineal, llamada función objetivo, la cual se encuentra sujeta a un conjunto de restricciones lineales. Para resolver este tipo de modelos destacan métodos como *branch and bound*, *branch and price*, *cutting-plane*, entre otros [22].
- **Conflict-Driven Clause Learning (CDCL):** Se usa para resolver el problema Boolean Satisfiability Problem (SAT) (dada una fórmula booleana, el problema le intenta encontrar una asignación de variables que haga verdadera la fórmula) propuesto por primera vez en [23]. Para su uso en este tipo de problemas, primero se debe reducir la instancia del problema en una formulación SAT, y luego encontrar una asignación de variables que satisfaga dicha formulación. Para mayor información consultar [24].

Por otro lado, se han estudiado ampliamente métodos *aproximados* donde destacan las meta-heurísticas. Éstas se centran en encontrar soluciones sub-óptimas dentro de un límite de tiempo. Para ello, en vez de realizar una búsqueda sobre todas las posibles configuración de variables (espacio de búsqueda), la realizan sobre ciertas áreas donde configuraciones óptimas son esperadas, ya que asumen, que éstas tienden a agruparse; así como también pueden compartir ciertas asignaciones de variables. Debido a lo anterior estos métodos tienen la particularidad de que son sensibles a su solución inicial, ya que de ella depende el siguiente subespacio de búsqueda. Se puede decir que estos métodos a diferencia de los exactos, que entregan la solución óptima, entregan la mejor solución encontrada durante su ejecución. Una extensa revisión del tema puede ser encontrada en [25]. Destacan técnicas como:

- **Búsqueda Local:** Se utiliza en problemas que pueden formularse como la búsqueda de una solución maximizando un criterio entre una serie de soluciones candidatas. Estos algoritmos se desplazan de una solución a otra en el espacio de las soluciones candidatas (el espacio de búsqueda) aplicando cambios locales, hasta que se encuentra una solución que se considera suficientemente buena o se ha cumplido algún otro criterio de término. Dentro de esta categoría se encuentran algoritmos como Búsqueda Tabú [26] y Simulated Annealing [27]. Más acerca de estos métodos puede ser encontrado en [28].
- **Colonia de Hormigas:** Es una técnica probabilística, dentro de los métodos de inteligencia de enjambres, que se usan para solucionar problemas que pueden reducirse a buscar los mejores caminos o rutas en grafos. Propuesto en [29], se basa en la imitación del comportamiento de las hormigas reales. En el algoritmo se emula cada proceso de rastro como lo es: la construcción del camino por medio de una feromona, la actualización de dicho camino, la evaporación y refuerzo del químico; simulando la rutina real de la hormiga para transporte.
- **Híbrido Evolutivo:** Propuesto en [30], combina dos o más algoritmos para resolver un mismo problema, con la finalidad de utilizar características destacadas de cada uno de ellos para mejorar el rendimiento del algoritmo (tiempo de convergencia) o la calidad de la solución. En simples palabras mezclan algoritmos locales (búsqueda tabú, recocido simulado) con algoritmos globales (genéticos).
- **Memético:** Propuestos en [31] requieren menos evaluaciones para encontrar óptimos locales e identifican soluciones de mayor calidad que los algoritmos evolutivos genéticos [32]. Este tipo de algoritmos añaden a la búsqueda global de un Algoritmo Genético

(GA) una optimización local de los individuos por medio de operadores conocidos como *memes*. Su idea central es realizar mejoras individuales de las soluciones en cada uno de los agentes junto con procesos de cooperación y competiciones de tipo poblacional.

Dentro del mundo de la optimización también se pueden encontrar algoritmos anytime [4], estos algoritmos entregan soluciones válidas a un problema incluso si son interrumpidos antes de su convergencia. De ellos es esperable, que ha medida que aumenta el tiempo de ejecución se encuentren mejores soluciones. Esto adquiere relevancia considerando que, para ciertas instancias del problema se puede requerir un tiempo extenso para encontrar soluciones y se puede contar con un tiempo limitado para la ejecución de un algoritmo. Si bien en este trabajo no se usan necesariamente algoritmos anytime, se observa el comportamiento anytime de los algoritmos utilizados.

2.2. Machine learning

Las técnicas de machine learning son un conjunto de técnicas que permiten detectar patrones en los datos y usar estos patrones para predecir comportamientos futuros o tomar decisiones bajo incertidumbre. Existen distintos tipos de técnicas de machine learning y pueden ser clasificadas en: *supervisados, no supervisados, semi supervisados y aprendizaje por refuerzo*. Tal como su nombre indica, en los métodos *supervisados* se tiene la observación y su etiqueta correspondiente, en un enfoque *no supervisado* solo se cuenta con las observaciones y no se tiene una etiqueta correspondiente a ella, el enfoque *semi supervisado* es una combinación de los mencionados anteriormente y finalmente el enfoque por *refuerzo*, requiere realizar una acción y evaluar el beneficio o castigo que trae realizarla [33].

En este trabajo se aplicara un enfoque *supervisado* donde destacan tareas como: *clasificación y regresión*.

2.2.1. Clasificación

Dado un conjunto de datos formado por un conjunto de observaciones que poseen un conjunto de atributos X con $X = \{x_1, x_2, \ldots, x_n\}$, donde cada observación se encuentra asociada a una clase c_m con m la cantidad de clases $y \in Y = \{c_1, c_2, \ldots, c_m\}$, se pretende

determinar, para cada nueva observación, su clase correspondiente. La clasificación puede ser binaria cuando $m = 2$, multi-clase cuando $m > 2$ y multi-etiqueta cuando una observación puede pertenecer a más de una clase simultáneamente.

Entre los modelos más utilizados para clasificación se encuentran los modelos basados en árboles (como *Decision trees* [34], *Random Forest* [35], *Gradient Boost* [36] and *Extra Trees* [37]) , *Support Vector Machine (SVM)* [38] y *Redes Neuronales* [39]. Algunas de las métricas utilizadas en clasificación puede ser encontradas en [40], sin embargo las utilizadas en este trabajo son:

- **Matriz de confusión**: Permite visualizar las predicciones entregadas por el modelo v/s las etiquetas reales. Las filas corresponden a las clases predichas por el modelo y las columnas corresponden a las clases verdaderas.

		Predicción	
		Positivos	**Negativos**
Observaciones	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Tabla 2.1: Matriz de confusión

De aquí es importante notar que se tienen cuatro casos posibles, como se muestra en la tabla 2.1.

- **Accuracy**: Corresponde al porcentaje de predicciones correctas. Esta métrica, tiene problemas cuando se considera un conjunto de datos, donde la variable objetivo se encuentra desbalanceada. En este caso, si el modelo siempre predice el valor con mayor frecuencia se obtendrá un alto accuracy.
- **Precisión**: Corresponde al porcentaje de las etiquetas correctamente predichas. Es una métrica que se calcula por clase.
- **Recall**: Corresponde a la efectividad de un clasificador para identificar correctamente las clases. Es una métrica que se calcula por clase.
- **Especificidad**: Corresponde a la efectividad de un clasificador para identificar correctamente un resultado negativo. Es una métrica que se calcula por clase.

- **F-Score**: Corresponde a una combinación de precisión y recall. En particular interesa el F1-score ($\beta = 1$), que corresponde a la media harmónica entre precisión y recall.

- **Cohen kappa** : Es una medida de confianza entre dos clasificadores sobre una misma observación, normalizado por la frecuencia con la que los clasificadores pueden estar de acuerdo al azar. Para calcularlo se necesita definir la probabilidad de acuerdo P_0 y la probabilidad de que estén de acuerdo al azar P_e que corresponde a la suma de la probabilidad $P_{acuerdo}$ más $P_{desacuerdo}$.

- **Coeficiente de correlación de Matthews (MCC)** : Esta métrica considera todos los valores presentes en la matriz de confusión. Su rango de valores esta entre -1 y 1; donde un valor de 0, indica que la predicción es aleatoria y un valor cercano a 1, indica que todas las clases se pueden predecir bien [41].

- **Media geométrica**: Esta métrica se ha definido de dos formas en la literatura la primera corresponde a la raíz de la multiplicación de la precisión y el *recall*, mientras que la segunda corresponde a la raíz de la multiplicación entre el *recall* y la *especificidad*. Para el desarrollo de esta tesis, nos quedamos con la segunda. Intenta maximizar el *accuracy* en cada clase mientras mantiene los demás balanceados. Esta definida entre 0 y 1, siendo 1 su mejor valor.[42]

- **Curva Receiver Operating Characteristic (ROC)**: Esta curva muestra el rendimiento de un clasificador con respecto al umbral de decisión. Si una observación pertenece o no a una determinada clase. Muestra el True Positive Rate (TPR): porcentaje de etiquetas correctamente clasificadas v/s False Positive Rate (FPR): el porcentaje de etiquetas incorrectamente clasificadas.

Las métricas que son calculadas por clases, puede ser micro o macro pesadas. La primera favorece las clase con mayor presencia en el conjunto de datos, mientras que la segunda trata a todas las clases por igual [40]. En la tabla 2.2 se muestra la fórmula de cada métrica.

Métrica	Fórmula
Accuracy	$\frac{VP+VN}{VP+FN+FP+VN}$
Precision	$\frac{VP}{VP+FP}$
Recall	$\frac{VP}{VP+FN}$
Especificidad	$\frac{TN}{TN+FP}$
F-Score	$(1+\beta^2)\cdot\frac{\text{precision}\cdot\text{recall}}{\beta^2\cdot\text{precision}+\text{recall}}$
Cohen Kappa	$P_0 = \frac{VP+VN}{VP+FN+FP+VN}$
	$P_{acuerdo} = (\frac{VP+FN}{VP+FN+FP+VN}) \times (\frac{VP+FP}{VP+FN+FP+VN})$
	$P_{desacuerdo} = (\frac{FP+VN}{VP+FN+FP+VN}) \times (\frac{FN+VN}{VP+FN+FP+VN})$
	$K = \frac{P_0+P_e}{1-P_e}$
MCC	$\text{MCC} = \frac{\text{VP}\times\text{VN}-\text{FP}\times\text{FN}}{\sqrt{(\text{VP+FP})(\text{Vp+FN})(\text{VN+FP})(\text{VN+FN})}}$
Geometric Mean	$GM_{PR} = \frac{TP}{\sqrt{(TP+FP)(TP+FN)}}$
	$GM_{SS} = \frac{\sqrt{TP\times TN}}{\sqrt{(TP+FN)(TN+FP)}}$

Tabla 2.2: Fórmulas métricas clasificación
VP, VN, FN, FP: definidos en la tabla 2.1

2.2.2. Regresión

A diferencia de la clasificación, donde se busca predecir una clase (valor discreto), en la regresión se busca predecir un valor continuo. Al igual que en el caso anterior, existe un conjunto X de variables, donde cada observación tiene un valor correspondiente $y \in \mathbb{R}$. Aquí se busca aproximar la función subyacente para estimar el valor y para nuevas observaciones [33].

Algunos de los modelos de regresión más usados pueden ser modelos lineales, como ElasticNet [43]. De la misma manera que se mencionaron modelos de árboles y redes neuronales en clasificación, estos modelos cuentan con su versión para realizar regresión. En cuanto a métricas, varias pueden ser encontradas en [44], aquí se definen las métricas utilizadas en este trabajo y en la tabla 2.3 sus respectivas fórmulas.

- **Root Mean Square Error (RMSE)**: Corresponde a la raíz cuadrada del promedio de la diferencia cuadrada entre el valor objetivo y el valor predicho por el modelo. Se prefiere en aquellos casos donde grandes errores no son deseados, ya que impone una

alta penalización a dichos errores.

- **Mean Absolute Error (MAE)**: Es la diferencia absoluta entre el valor objetivo y el valor predicho, es una métrica robusta a valores extremos y no penaliza tanto los errores como RMSE.
- **Mean Absolute Percentage Error (MAPE)**: Es la versión porcentual de MAE, corresponde a qué tan lejos se encuentran las predicciones del modelo con respecto al valor real en promedio. A pesar de ser una medida robusta a valores extremos, tiene el problema de que cuando los valores predichos tienden a ser menores que el valor actual, la métrica entregará un menor valor, en contraste a lo que sucedería si el valor es mayor en las mismas cantidades.

Métrica	Fórmula
RMSE	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$
MAE	$\frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$
MAPE	$\frac{100\,\%}{N}\sum\left[\frac{y-\widehat{y}}{y}\right]$

Tabla 2.3: Fórmulas métricas regresión
y_i valor real, $\hat{y}_i$ valor predicho y N cantidad de muestras

2.3. Validación cruzada

La validación cruzada es una técnica que se utiliza para evaluar los resultados obtenidos por un modelo y garantizar que los datos de entrenamiento y de evaluación son independientes entre sí [45]. Existen distintas formas de hacer validaciones cruzadas, las relevantes para este trabajo son:

- *K-folds:* Consiste en realizar un número k de iteraciones, para los cuales los datos son divididos en k subconjuntos, en cada iteración se entrena el modelo utilizando $k-1$ subconjuntos, haciendo variar en cada iteración el subconjunto que es utilizado para evaluar el modelo.
- *LOOCV (Leave One Out-CV):* Aquí en vez de utilizar grupo de observaciones, solo una observación es descartada del conjunto de entrenamiento y utilizada para evaluar el modelo.

- *Stratified:* Este indica que la separación entre los conjuntos se hace manteniendo la proporción de los valores objetivos en ambos.
- *Group:* Separa las instancias haciendo que todas las observaciones correspondientes a un grupo de instancias se encuentre totalmente contenido en un conjunto o en otro.

Para este trabajo se utilizara una combinación de varios tipos denominada *stratify-group-k-folds* que agrupa las propiedades de los tipos anteriormente mencionados.

2.4. Selección automática de algoritmos

Una extensa revisión sobre selección automática de algoritmos puede ser encontrada en [46, 47]. Fue propuesta por Rice en [3], donde trata de responder a la pregunta: *Dados diferentes algoritmos para resolver un mismo problema, ¿cuál de los algoritmos debe ser seleccionado para resolver una instancia en particular?*. Para entregar una respuesta se identificaron cuatro componentes importantes:

- El conjunto candidato de algoritmos A.
- Las instancias del problema y el espacio del problema P.
- Atributos medibles de una instancia y el espacio de las características denotado como F.
- El espacio de rendimiento Y.

Lo anterior puede ser resumido en la Figura 2.1.

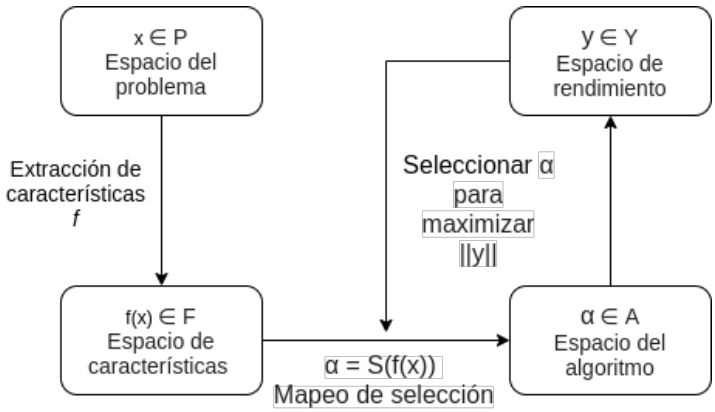


Fig. 2.1: Marco de trabajo de *Rice* para la selección de algoritmos [3].

Formalmente, el problema puede ser descrito como: Para una instancia dada del problema $x \in P$, con características $f(x) \in F$ encontrar el mapeo de selección $S(f(x))$ en el espacio del algoritmo A, tal que el algoritmo seleccionado $\alpha \in A$, minimice el mapeo del rendimiento $y(\alpha(x)) \in Y$. De lo anterior el conjunto de A,P,S,Y se conoce como *meta data*.

La selección de algoritmos está estrechamente relacionada con la idea de portafolio de algoritmos [48] y el diseño de éstos. La gran diferencia es que originalmente los portafolios ejecutan los algoritmos de manera paralela (concurrente), en contraste con la selección de algoritmos que ejecutan el que obtiene el mejor resultado. En algunas variaciones de portafolios los algoritmos pueden compartir información entre ellos, esto sucede en los portafolios secuenciales (uno por uno) o de ejecución parcial [49], esto permite combinar algoritmos de manera dinámica, de esta manera se podría obtener mejores resultados en comparación a usar un algoritmo.

Una de las partes fundamentales para la selección automática de algoritmos, es la colección de algoritmos disponibles a utilizar, el cual puede ser infinito; además es posible mezclar métodos exactos y aproximados, causando que existan algoritmos eficientes o no, ya que, en lo que corresponde a una selección óptima no se seleccionaran algoritmos sub-óptimos. Es por esto que identificar un conjunto de algoritmos acotados y que destaquen del promedio, requiere de un gran conocimiento del problema en particular.

Además de la colección de algoritmos a usar, es necesario considerar las características que permiten caracterizar las instancias y que adquieren gran importancia. Actualmente no hay una forma automática de encontrarlas, por ende, se necesita dominio y habilidades analíticas del tema. En la literatura se han propuesto características generales [50] que pueden ser utilizadas de acuerdo al tipo de estructura que se use para modelar el problema. Se pueden considerar ciertas recomendaciones, como por ejemplo: "toda característica debe ser creada solamente usando la instancia y sin depender de ninguna otra información". Otra recomendación esta relacionada con el problema de *metareasoning-partition* [51], donde mientras más tiempo es requerido para realizar el *metareasoning*, existe un menor tiempo para ejecutar la instancia y resolver el problema en sí. También se debe tomar en consideración que las características no estén fuertemente correlacionadas entre sí, ya que juntas, pueden no entregar mayor información, sino añadir ruido y aumentar la dimensionalidad del problema.

También debe ser considerado el espacio de las instancias ya que son críticas para definir

el espacio del problema. Para ello se debe contar con un amplio espectro de ellas, éstas pueden ser recolectadas desde problemas reales o pueden ser obtenidas desde generadores de instancias. Se debe intentar conseguir instancias que sean representativas a todas las clases de instancias posibles para un problema.

Para decidir qué algoritmo es mejor que otro, se podrían utilizar herramientas de teoría de la complejidad, específicamente el análisis del comportamiento asintótico de los algoritmos y escoger el caso "peor-mejor-promedio" de un algoritmo para realizar comparación entre ellos. El problema radica en que algunos algoritmos puedan mostrar comportamientos asintóticos similares, aunque en la práctica esto no sea así. El rendimiento puede depender de la implementación o del rendimiento de una instancia particular.

Para evitar los tipos de errores anteriormente mencionados, es que puede ser útil hacer uso de los resultados experimentales de la ejecución de éstos, sobre distintas instancias para comparar los resultados, definiendo tiempos límites de ejecución o cantidad de restricciones chequeadas como condiciones de término de las ejecuciones. En este contexto, donde lo que se quiere predecir es qué algoritmo es el más adecuado para resolver una instancia del problema, el *machine learning* puede ayudar a responder esta duda; usando principalmente técnicas como *clasificación* (en sección 2.2.1) y *regresión* (en sección 2.2.2) En estos modelos se busca que aprender el mapeo de selección de rendimiento de las instancias del problema usando las características extraídas. Esto permite crear un modelo que logre aprender el rendimiento del algoritmo para posteriormente usarlo y predecir sobre nuevas instancias.

3. ESTADO DEL ARTE

En esta sección se presentan artículos y metodologías encontradas en la literatura, que aportan algún concepto o idea a este trabajo. Los trabajos revisados son los publicados hasta el año 2019. La revisión del estado del arte se encuentra divida en las siguientes partes:

3.1. Coloreo de grafos

3.1.1. Solvers

A continuación se detallan los solvers encontrados en la literatura que fueron considerados para el desarrollo de esta tesis.

- **Degree Saturation (DSATUR)**: Propuesto por Daniel Brélaz [52], es un algoritmo greedy que colorea los vértices uno a uno. Cada vez que un vértice es coloreado los vértices son ordenados usando la heurística de *Degree Saturation* que corresponde a ordenar los vértices faltantes según la mayor cantidad de colores asignados a sus vecinos.
- **Recursive Largest First (RLF)**: Propuesto por Leighton [53], es un algoritmo greedy que produce un coloreo a la vez. En cada paso, el algoritmo usa una heurística para identificar el conjunto independiente de vértices, que se pueden asociar a un mismo color, este conjunto independiente es quitado del grafo. El proceso es repetido hasta que se obtiene un subgrafo vacío.
- **Tabucol**: Propuesto en [54], busca en los coloreos impropios para disminuir el número de asignación de colores que hacen infactible la solución, conocido como choque. Dada una solución candidata S se analizan los vértices que causan los choques y a cada uno de estos se le busca asignar un nuevo color de tal manera que no produzca choques y consecuentemente disminuya la cantidad de éstos.
- **GC-Ant Colony**: Propuesta por Dowsland and Thompson [55] combina operadores de búsqueda global y local. Basado en la meta-heurística de Colonia de Hormigas (ver sección 2.1.3). Utiliza hormigas para producir soluciones candidatas individuales. Durante cada ejecución cada hormiga produce una solución de manera no-determinista, usando

probabilidades basadas en heurísticas y calidad de soluciones generadas por otras hormigas. Si hormigas anteriores detectan características que llevan a soluciones mejores que el promedio, las futuras son más propensas a considerar estas características en su propia solución, lo que generalmente lleva a una reducción de colores en esa ejecución. Las hormigas producen soluciones completas que no necesariamente son factibles.

- **Backtracking DSATUR**: Propuesto en [56], opera de manera similar al algoritmo de Backtracking (ver sección 2.1.3), utilizando: el orden inicial de los vértices es determinado por DSATUR. Así, los vértices con una menor cantidad de colores posibles son coloreados primero, los empates se rompen por los grados de los vértices y empates posteriores de manera aleatoria. También, luego de realizar un paso de retroceso, los vértices son ordenados usando la cantidad de colores disponibles. En el caso de que encuentre vértices que no tienen colores disponibles, el algoritmo retrocede un paso.

- **Color 6**: Propuesto en [57], es un algoritmo exacto basado en la idea de un algoritmo de Backtracking (ver sección 2.1.3), *backGCP* aplicando CDCL para descubrir restricciones implícitas sobre los vértices durante la búsqueda y podar el espacio de búsqueda. En cada nodo del árbol de búsqueda, intenta expandir la solución actual A añadiendo nuevos vértices coloreados. Para recorrer el árbol, usa tres procedimientos: *UnitPropagation*, *AnalyzeConflict*, y *Backtracking*. De esta manera se reduce C_v, que es el conjunto de colores disponibles para un nodo sin colorear, en el caso de que C_v sea de tamaño unitario, se aplica *UnitPropagation*, ya que, solo existe una sola posibilidad de satisfacer la cláusula. Por otro lado, cuando C_v esta vacío, se debe aplicar *AnalyzeConflict* para determinar qué cláusula llevo a un conflicto; de esta manera, esta cláusula es almacenada y añadida al conjunto de restricciones que ayudan a reducir el espacio de búsqueda. De esta manera, el algoritmo recorre el espacio de las soluciones sistemáticamente.

- **GC-CDCL**: Propuesto en [58] es un enfoque híbrido entre CP/SAT, usando el algoritmo CDCL con un encoding basado en los *Árboles de Zykov* donde para dos nodos no vecinos, puede darse que toman colores distintos entre ellos y son unidos por una arista o toman el mismo color y pueden ser fusionados en uno solo. Si se ramifica cuando dos vecinos obtienen el mismo color, se obtiene un árbol sin simetría con grafos completos como hojas. De esta manera, el número cromático de cada grafo hoja es una cota inferior del problema. Además proponen un método para producir una explicación clausal para usar en CDCL y una heurística de ramificación para simular el solver DSATUR sobre los

Árboles de Zykov.

- **Picasso**: Propuesto en [59], es un algoritmo basado en CDCL (ver sección 2.1.3), en él se propone una nueva forma de realizar un encoding a Conjunctive Normal Form (CNF) desde el grafo basado en el *Árbol de Zykov*. Usando la codificación anterior, ellos proponen usar Counter-Example Guided Abstraction Refinement (CEGAR), esto permite codificar una parte del problema e ir añadiendo restricciones faltantes de manera incremental hasta decidir la satisfacibilidad del problema.

- **GC Hill-Climbing** Propuesto en [60], opera en el espacio de las soluciones factibles, con una solución inicial generada por DSATUR . En una ejecución el algoritmo opera en una solución factible S con el objetivo de minimizar el número de colores en la solución $|S|$. Primero un pequeño número de colores es removido desde S y llevado a un conjunto T, dando dos soluciones parciales. Un algoritmo de búsqueda local especializado es ejecutado por I iteraciones, para intentar transferir vértices desde T a S, de tal manera que ambos conjuntos mantengan soluciones factibles. En el caso de funcionar se aumenta la cardinalidad de S, disminuyendo la cantidad de colores en la solución. Al final de este proceso, todas las clases de colores en T son copiadas devuelta a S para formar una solución factible.

- **Hybrid Evolutionary Algorithm (HEA)** Propuesto en [61]. Este método es un algoritmo genético que usa un enfoque de partición, trabajando en coloreos completos pero ilegales. Utiliza búsqueda local en reemplazo de un operador de mutación y un operador de recombinación especializado denominado Greedy Partition Crossover (GPX). Éste intenta construir una descendencia usando grandes clases de colores heredadas de ambas soluciones padres. El algoritmo crea una población inicial de soluciones candidatas, éstas son creadas usando una versión modificada de DSATUR con un número de k colores definido desde el inicio, para generar una diversificación en la población. El primer vértice es seleccionado al azar y se le asigna el primer color, luego, los vértices son seleccionados de acuerdo al grado de saturación máxima asignándole el color que tiene una menor cantidad de vértices asignados. Cuando se encuentra con una asignación no factible se intenta asignar otro color de manera aleatoria. En cada iteración, dos soluciones padres son seleccionadas al azar y se pasan por un operador de recombinación para producir una solución hijo, la cual es mejorada por un operador local e insertada en la población reemplazando al padre más débil.

- **MACOL**: Propuesto en [62], es un algoritmo memetico (ver sección 2.1.3). Este integra la búsqueda tabú con un algoritmo evolutivo. En él se propone un nuevo operador adaptativo de recombinación que es una extensión del GPX, denominada Adaptive Multi-Parent crossove (AMPaX) y un nuevo criterio de reemplazo para la actualización de la población que considera la calidad de la solución y la diversidad de los individuos. AMPaX usa 2 o más padres para generar una descendencia y en cada paso de recombinación escoge un padre y un color para transmitir a su descendencia. Para el criterio de reemplazo se mide la similitud entre las soluciones padres y la solución hijo. Esta similitud puede ser una métrica de distancia entre dos coloreos posibles y la distancia entre un coloreo y una población. De esta manera, si la descendencia es similar a algún padre o alguna solución ya considerada, la descendencia será descartada.

- **Hybrid Evolutionary in Duet** Propuesto en [63], esta basado en HEA, con la diferencia que trabaja con una población de 2 individuos y una nueva forma de manejar diversidad. El algoritmo puede ser visto como dos algoritmos *Tabucol* que interactúan mediante crossover. El parámetro principal del solver es la cantidad de iteraciones a realizar por *Tabucol*, los parámetros de éste y la cantidad de ciclos.

  Luego de inicializar aleatoriamente dos soluciones c_1 y c_2 para generar diversidad y dos soluciones elite e_1 y e_2, el algoritmo es repetido hasta que se llega a un criterio de término, que pasa cuando no se encuentran conflictos en la mejor solución o si las soluciones encontradas por Tabucol p_1 y p_2 son iguales.

  Primeramente se introduce diversidad a las soluciones c_1 y c_2, con el operador de crossover, luego, las soluciones son mejoradas mediante el algoritmo de Tabucol, se guarda la mejor solución y los padres son reemplazados por los hijos, una iteración de esto se llama una generación. Luego de un número de generaciones, las soluciones c_1 y c_2, se vuelven similares en el espacio de búsqueda. Para mantener la diversidad, se reemplaza una de las soluciones por una solución previamente encontrada por el solver; haciendo a e_1 la mejor solución actual y e_2 es la mejor solución del ciclo anterior. Al final de cada ciclo e_2 reemplaza a p_1 o a p_2.

- **Partialcol** Propuesto en [64], es un prototipo de dos heurísticas que usan una lista tabú. Este algoritmo trabaja con coloreos factibles, pero incompletos. Una solución actual es representada por k clases de colores y un conjunto de nodos no coloreados. Durante la búsqueda, intenta reducir el conjunto de nodos no coloreados a 0, usando como función

objetivo el tamaño de conjunto sin colorear. Una vecindad es definida moviendo un nodo u desde el conjunto sin colorear para asignarle un color. En el caso de que la solución sea infactible, los nodos adyacentes a u son movidos a los nodos sin colorear, hasta que sea una solución legal nuevamente. En cada iteración, se escoge el nodo u de tal manera que la cantidad de nodos sin colorear sea mínima.

- **Hybrid partial-ordering based ILP (pop2)** Propuesto en [65], es un algoritmo exacto que mezcla la formulación del coloreo de grafos, basado en asignación, con la idea del *partial-ordering problem* (pop). Asumiendo que los k colores están ordenados linealmente, se debe determinar un orden relativo de cada vértice con respecto a cada color en el ordenamiento de colores. De esta manera, para cada color i y para cada vértice $v \in V$; las nuevas restricciones añadidas a la formulación de asignación entregan información si v es menor/mayor que i. Así se tiene que cada par (v, i) es comparable. Con esta idea ellos proponen un modelo que obtiene una menor cantidad de restricciones que la formulación de asignación.

3.1.2. Instancias

Como se mencionó anteriormente, se trabajará con instancias públicas y generadas.

3.1.2.1 Generadores de instancias

Son algoritmos que permiten, bajo ciertos parámetros, generar distintas instancias de un problema en específico. Para este trabajo son relevantes:

1. **Networkx Generator:** Este conjunto de instancias fue creado utilizando la librería Networkx [66] en python. Esta librería permite crear una amplia variedad de grafos. Para cada tipo de grafo se hicieron variar atributos como: n cantidad de vértices, m cantidad de aristas, $seed$ semilla del generador, d cardinalidad de regularidad del grafo, $prob$ la definición de esta variable depende del tipo utilizado, k la cantidad promedio de vecinos con los que se conecta cada vértice. A continuación se mencionan los distintos tipos de grafos encontrados en trabajos revisados:

- ***Watts strogatz, Newman watts strogatz, Connected watts strogatz*:** Son grafos de *Red de mundo pequeño*. Para este tipo de grafo, la mayoría de los nodos no son vecinos entre sí; sin embargo, la mayoría de los nodos pueden ser alcanzados desde cualquier nodo origen a través de un número relativamente corto de nodos entre ellos. Para generarlo se define un anillo entre todos los nodos y luego cada nodo es conectado a k vecinos más cercanos; entonces se crean atajos entre nodos siguiendo: para cada par de arco (u, v) en el subyacente "n-anillo con k vecinos más cercano" con una probabilidad $prob$ de reemplazarlo por un arco de (u, w) con una elección uniformemente aleatoria del nodo w existente.

  En *Newman watts strogatz* la reconexión aleatoria puede aumentar el número de bordes. Además, la reconexión no siempre asegura tener un grafo conexo, excepto en el caso de *Connected watts strogatz*.

- ***Duplication divergence:*** Se crea un grafo duplicando los nodos iniciales y mantiene aristas que inciden con los nodos originales con una probabilidad de retención $prob$.

- ***Partial duplication graph:*** En una primera instancia, se crea un grafo completo de tamaño n, luego se repiten los siguientes pasos hasta que N nodos son alcanzados: desde un nodo u se crea un nuevo nodo v, para cada vecino de u; se crea una arista hacia v con una probabilidad $prob$. Finalmente, una arista entre u y v es creado con una probabilidad q. Estos grafos son inspirados en procesos biológicos.

- ***Gnm random y Dense gnm random:*** Estas clases de generadores sirven para generar el mismo tipo de grafo. Sin embargo, los generadores se diferencian en que usan distintos algoritmos para su generación. En *Gnm random* se ocupa un algoritmo probabilístico y los grafos resultantes son conocidos como grafos binomiales, mientras en *Dense gnm random* ocupan una implementación que funciona mejor con grafos densos.

- ***Random regular:*** Retorna una grafo d-regular, es decir, todos los nodos del grafo tienen d grados.

2. **Joe Cuberlson Generator**[1]: En palabras de su creador "Mi intención es proporcionar varios generadores de grafos que apoyarán la investigación empírica de las características de varios algoritmos de coloración. Por lo tanto, quiero generadores que exhiban variacio-

[1]Disponible en: `http://webdocs.cs.ualberta.ca/~joe/Coloring/`

nes sobre distintas características de los grafos, como el grado (expectativa y variación), coloreos ocultos, cintura, distribución de los bordes, etc." Los tipos de grafos que pueden ser generados basados en la distribución de las aristas pueden clasificarse en:

- ***Uniformes:*** En este tipo, las aristas se asignan a un par de nodos con una probabilidad fija de p.
- ***Geométricos:*** Estos son creados localizando uniformemente los nodos en un cuadrado bidimensional y asignando un borde entre 2 nodos. Si la distancia euclidiana es menor o igual a un parámetro r.
- ***Peso-sesgados:*** Primeramente, los nodos son asignados a conjuntos de nodos independientes; luego, a cada par de nodos se asigna un peso y las aristas son creadas de manera iterativa con una probabilidad proporcional al peso del par de nodos. Cuando se genera una arista nueva, los pesos son reducidos. Se evita la generación de aristas entre nodos de conjuntos diferentes.
- ***Guiados por ciclos:*** Crea grafos con ciclos de una longitud determinada. Se crean particiones de k-color, luego genera un ciclo generando aleatoriamente un camino con cada vértice perteneciente a una partición de color diferente a la última.
- ***Grado de nodos y cintura inhibidos:*** La cintura corresponde a la longitud del ciclo más corto contenido en un grafo. A cada grafo se le asigna una probabilidad p, una cintura límite g, y un grado límite de nodos Δ. El límite de la cintura indica que no se creará ningún ciclo con una cintura inferior a g . Por lo tanto, si se considera un borde (v, w) como un nuevo borde, cada par de vértices (x, y) debe tener una distancia inferior a g. p es la probabilidad de que se utilice un posible borde. Δ es un límite duro en la diferencia entre el grado medio de nodo y el grado máximo de cualquier vértice.

3. **BHOSLIB**[2]: Las instancias disponibles en este conjunto corresponden a las generadas en el modelo RB propuesto en [67]. Este generador se basa en el conocimiento de reglas para problemas SAT y CSP, entre otros. Permiten generar instancias difíciles variando parámetros de control sobre valores críticos que pueden ser calculados. Para esto se genera una red de restricciones con un conjunto de variables finito, donde cada variable X está asociado a un dominio de posibles valores y un conjunto finito de restricciones C, que representa una relación que figura al conjunto de tuplas permitidas para las

[2]Disponible en `http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`

variables en C. Una solución para la red corresponde a una asignación de valores válidos a la misma, tal que, todas las restricción sean satisfechas. Una red de restricciones es *satisfacible* si admite al menos una solución. En específico, el modelo RB, se caracteriza porque permite seleccionar restricciones con repetición, y porque el tamaño del dominio de cada variable crece polinomialmente con respecto al número de variables.

4. **Hisampler**[3]: [68] Es un algoritmo basado en aprendizaje por refuerzo, que permite generar instancias difíciles para un solver proporcionado como entrada. Este trabajo esta enfocado principalmente sobre problemas sobre grafos. Es un generador probabilístico de instancias difíciles. Una vez que la distribución generativa de instancias difíciles es obtenida para un solver, se puede muestrear instancias que permitan generar un conjunto de benchmark. A diferencias de otros trabajos, se busca modelar distribuciones que funcionen bien sobre casos generales, más que estar basado en reglas específicas de un problema. Experimentalmente se encontró que puede generar grafos no conexos.

3.1.2.2 Instancias públicas

1. **Dimacs**[4]: Posee 136 instancias del desafío *COLOR02/03/04: Graph Coloring and its Generalizations* [69]. Este conjunto de datos es emblemático para el problema de coloreo de grafos, que es usado como benchmark para distintos solvers. Reúne instancias de distintos problemas propuestos por distintos autores. Éstos han sido ocupados para evaluar distintos modelos. Una descripción detallada de las instancias puede ser encontrada en el trabajo [69].

2. **GNP**[5]: Usado en el paper [65], consiste de 340 grafos generados de forma aleatoria usando el generador networkx. Para cada grafo $G(n, p)$ con n vértices, se genera una arista entre cada par de vértices con un probabilidad p. En este trabajo se usaron parámetros diferentes a los usados anteriormente, los autores los dividen en dos:

    - ***set100:*** Contiene 100 instancias con 70 vértices, son 20 instancias para cada p =0.1, 0.3, 0.5, 0.7, 0.9.

[3]Disponible en `https://github.com/joisino/HiSampler`
[4]Disponible en `https://mat.tepper.cmu.edu/COLOR02/`
[5]Disponible en `https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks`

- ***sparse240:*** Contiene 240 instancias de 80, 90 y 100 vértices, son 20 instancias para cada $p = 0.1, 0.15, 0.2, 0.25$.

3. **Chi**[6]: El conjunto de instancias chi500 y chi1000, son proporcionados por Marco Chiarandini y Thomas Stützle [70] y contiene 520 instancias con 500 vértices y 740 instancias con 1000 vértices respectivamente. Estas instancias son creadas usando el generador de Culberson [71]. Tal como se menciona en el paper [70], el dataset se generó controlando parámetros como la densidad de los arcos o la distribución de éstos. La variable p determina la proporción de aristas entre los vértices. Para este conjunto se consideran los valores $p = 0.1, 0.5, 0.9$, utilizando 3 clases de grafos de los 5 disponibles que corresponden a uniformes, geométricos y pesos sesgados.

4. **Sports Scheduling**[7]: Este conjunto corresponde a 10 instancias que representan torneos round robin de las ligas deportivas [72] como instancias de coloreo de grafos, para esto se entrega un número par de n equipos y cada equipo debe participar en un encuentro contra todos los demás equipos m veces en $m(n-1)$ rounds. El proceso para transformar desde un problema a otro es descrito con más detalle en [73].

5. **Hugo Hernandez**[8]: Este conjunto de instancias, disponible en [74], corresponde a instancias recolectadas en la página de Hugo Pibernat. En ella se agrupan instancias de distintas fuentes. Y consiste de un total de 238 instancias. De estas 90 fueron seleccionadas, debido a que las demás ya fueron consideradas en otros conjuntos.

3.2. Selección automática de algoritmos para coloreo de grafos

A continuación se describirá el estado del arte, utilizando el enfoque de Rice (Sección 2.4), enfocado en los trabajos realizados para GCP.

En [75] se propone una extensión a la metodología propuesta por Rice. Allí se busca definir el *footprint* [76] de un algoritmo, que corresponde a la región generalizada donde un algoritmo en específico se espera que obtenga un buen rendimiento. En él se propone un marco de trabajo que permite ver cómo las fortalezas y debilidades relativas de cada algoritmo pueden

[6]Disponible en `https://imada.sdu.dk/~marco/gcp-study/`
[7]Disponible en `http://www.rhydlewis.eu/`
[8]Disponible en `https://www.cs.upc.edu/~hhernandez/graphcoloring/`

ser visualizadas y medidas objetivamente, además de cómo las propiedades de las instancias pueden ser usadas para predecir el rendimiento del algoritmo en instancias no vistas previamente con alta precisión. Esta metodología puede ser resumida en la Fig. 3.2. Esta nueva técnica de trabajo propuesta combina la idea de Rice con el uso de *footprints*, haciendo necesario una reducción de dimensionalidad en el espacio de características.

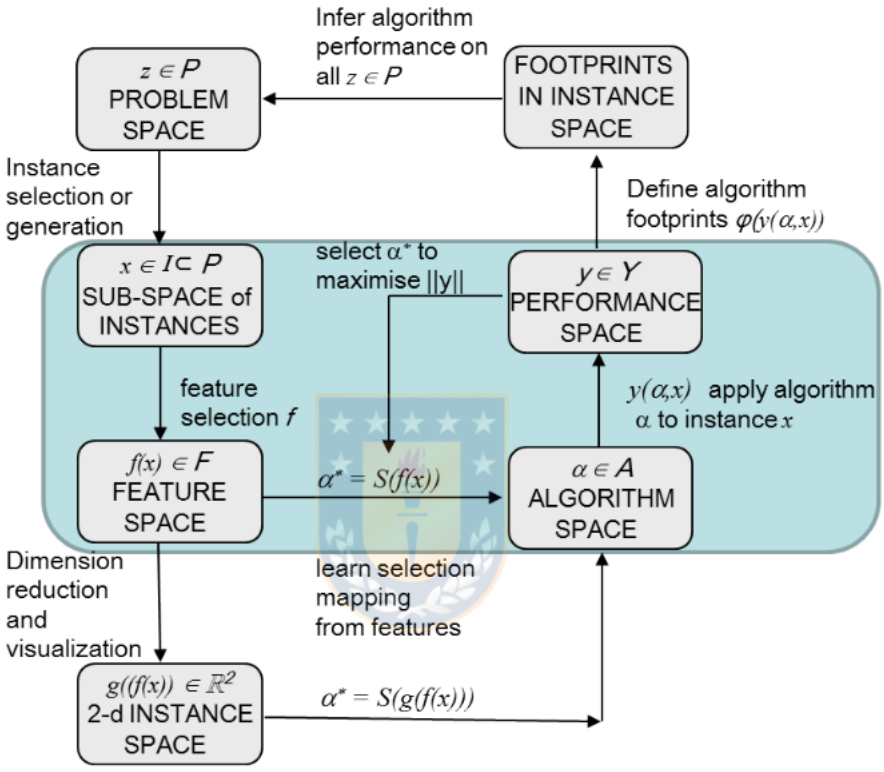


Fig. 3.2: Marco de trabajo extendido* [75]

*Esta metodología mantiene la estructura de Rice, pero es extendida ya que considera como espacio del problema todas las posibles instancias del problema, además de una selección de características y una reducción de dimensionalidad a un espacio 2 dimensiones común para los algoritmos, con el cual, se puede visualizar y medir el poder relativo de ellos. Además de definir las regiones donde se espera que los algoritmos obtengan un buen rendimiento, denominadas *footprints*.

Realizando un análisis por módulos en la revisión bibliográfica, se tiene que:

3.2.1. Instancias

En la bibliografía es posible encontrar trabajos que realizan sus estudios con conjuntos de instancias, que van desde 101 hasta aproximadamente 7000 instancias, obtenidas desde dataset clásicos como dimacs [69] o de generadores como networkx [66] utilizado para crear solo grafos aleatorios y geométricos, y Culberson's [71] donde existen 5 clases de grafos. Es importante mencionar que encontrar instancias basadas en problemas reales se hace complicado; por esto, varios trabajos generan una gran cantidad de instancias para estudiar distintos métodos, aunque los generadores no necesariamente pueden considerar la totalidad de clases de instancias que existen para el problema.

3.2.2. Características

Respecto a las características de los trabajos, éstos consideran un número distinto y principalmente con valores numéricos sobre los valores categóricos. En particular, de [77] se destaca la incorporación de estadísticos como: la media, la desviación estándar, los quintiles, el coeficiente de variación, que indica qué tanto varían los valores con respecto a la media y la entropía, que indica la cantidad de información contenida en los datos. Además también se consideran características que corresponden a secuencia de valores. Por su lado, en [78], se destaca el análisis enfocado principalmente en las características espectrales de los grafos.

Entre las características más relevantes, propuestas por los trabajos anteriores, se encuentran:

1. El **número de vértices** en un grafo $n = |V|$.
2. El **número de aristas** en un grafo $|E|$.
3. **Relaciones entre aristas y vértices** $\frac{n}{m}$ y $\frac{m}{n}$.
4. **La densidad de un grafo:** la relación entre el número de aristas y el número de vértices. $\rho = \frac{2|E|}{n(n-1)}$

5. **Longitud de los caminos**: el número medio de pasos a lo largo de los caminos más cortos para todos los pares de vértices posibles. Además, refleja la eficiencia de viajar entre los vértices.

6. **Grados de los nodos:** Corresponde a la cantidad de nodos adyacentes de un nodo en particular u, y, en este caso, es una medida de que tan restringido esta u. Esta característica es calculada para cada nodo en V.

7. **Coeficiente de clustering:** Es una medida del grado en el que los nodos de un grafo tienden a agruparse. Para definirlo es necesario saber que es una tripleta (tres nodos conectados por 2 aristas como en la Fig. 3.3a, y esta es cerrada cuando los nodos están conectados por tres aristas como en la Fig. 3.3b). Existen tres tipos de coeficientes:

   - *Coeficiente de clustering global:* Propuesto en [79], corresponde a la relación entre el número de tripletas cerradas y el número total de tripletas con c definido como:

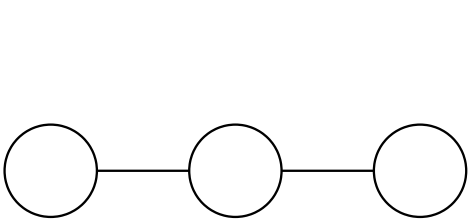
   $$c = \frac{\text{número de tripletas cerradas}}{\text{número de tripletas}} \tag{2}$$

   - *Coeficiente de clustering local:* Propuesto en [80], el coeficiente de clustering de un nodo u indica cuán fuertemente conectado se encuentra u con sus vecinos y qué tan cerca están de formar una clique. Es la proporción del número de aristas entre sus vecinos y el número máximo de aristas posibles entre ellos. Si u tiene d_u vecinos, el número de aristas entre ellos esta dado por e_u, definido como:

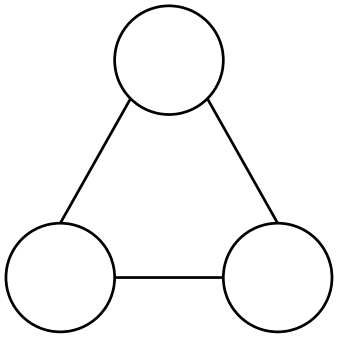
   $$e_u = |\{(v, v') : (u, v) \in E, (u, v') \in E, (v, v') \in E\}| \tag{3}$$

   De esta manera, el coeficiente de clustering local C_u esta definido como $C_u = \frac{e_u}{d_u \cdot (d_u - 1)}$.

   - *Coeficiente de clustering pesado:* Según lo propuesto en [77], se puede ponderar el coeficiente de clustering de un nodo por su grado, de esta manera se puede incorporar información acerca del vecindario de un nodo en específico.



(a) Tripleta



(b) Tripleta cerrada

Fig. 3.3: Tripletas

8. **Soluciones de solvers greedy:** Se utilizaron solvers como DSATUR y RLF (ver sección 3.1.1), de ellos se almacena el número de colores utilizado por cada solver, los ratios entre soluciones $\frac{k_{DSATUR}}{k_{RLF}}$, $\frac{k_{RLF}}{k_{DSATUR}}$ y los tamaños de los conjuntos independientes obtenidos por el solver que obtiene una mejor solución. El uso de estas soluciones como característica, puede ser considerado como un límite superior para una instancia en particular.

9. **Betweenness centrality:** Propuesto en [81] representa qué tan central es un nodo con respecto al grafo. Si se considera un nodo u, se define como el número de los caminos más cortos desde todos los nodos a todos los nodos que pasan por u. Con $\sigma_{s,t}$ el número de caminos más cortos entre los nodos s, t y $\sigma_{s,t}(u)$, son los caminos que incluyen a u. La betweenness centrality $g(u)$ se encuentra dada por $g(u) = \sum \frac{\sigma_{s,t}(u)}{\sigma_{s,t}} \mid s,t \in V, s \neq u \neq t$.

10. **Excentricidad**: Propuesto en [82], considera la mayor distancia entre un nodo y cualquier otro nodo presente en el grafo. La distancia es la cantidad de aristas que están en el camino más corto entre nodos. De aquí, la menor distancia entre todos los nodos del grado se conoce como radio de un grafo y la mayor distancia como diámetro.

11. **Descomposición de ancho de árbol:** Corresponde al mapeo de un grafo $G = (V, E)$ a una estructura de árbol T. En él, cada nodo $t \in T$, representa un subconjunto de los vértices del grafo $V_t \subseteq V$, de cada subconjunto se debe cumplir que:

    - Por cada vértice $v \in V$ existe algún V_t tal que $v \in V_t$.
    - Para todas las aristas $(u, v) \in E$ existe algún V_t que incluye a u y v.
    - Para cualquier conjunto V_{t_1}, V_{t_2} de los nodos $t_1, t_2 \in T$ ambos contienen un nodo $v \in G(v \in V_{t_1} \wedge v \in V_{t_2})$ y cualquier tercer nodo $t_3 \in T$ esta en el camino desde

t_1 a t_2, el nodo v también esta incluido en V_{t_3}.

De aquí, el ancho de una descomposición de árbol, es el tamaño del conjunto más grande en V_t menos uno y treewidth de un grafo G es el mínimo ancho sobre todas las posibles descomposiciones de G.

12. **Características asociadas a la matriz de adyacencia:**

    - Energía: Propuesto en [83], corresponde a la media de los valores propios de la matriz de adyacencia.
    - El primer y segundo valor más pequeño de los valores propios de la matriz de adyacencia.
    - El primer y segundo valor más grande de los valores propios de la matriz de adyacencia.
    - Diferencia entre el mayor y el segundo mayor valor propio de la matriz de adyacencia.

13. **Características asociadas a la matriz de Laplace:**

    - Conectividad algebraica: Propuesto en [84], es el segundo menor valor propio de la matriz de Laplace y refleja qué tan bien conectado se encuentra un grafo.
    - El primer y segundo valor más pequeño, distinto de cero, de los valores propios de la matriz de Laplace.
    - El primer y segundo valor más grande de los valores propios de la matriz de Laplace.
    - Diferencia entre el mayor y el menor, distintos de cero, valor propio de la matriz de Laplace.

3.2.3. Solvers usados en otros trabajos de selección de algoritmos

Los solvers utilizados en [78, 85, 86], no corresponden al estado del arte de los años de publicación de los papers revisados; sin embargo, ayudan a tener la conceptualización e ideas para abordar el problema. Estos trabajos no son tan útiles como otros, ya que la cantidad de solvers utilizados es acotada y de una naturaleza similar. Por el contrario, en [75, 77] se evalúan varios solvers de distinta naturaleza, de los cuales, algunos actualmente pueden ser considerados como estado del arte en sus respectivas técnicas como AntCol [55], branch & bound [87],

Dsatur [52], HybridEA [61], ILS [88], MAFS [89], MMT [90], FOO-PARTIALCOL [64], Tabucol [54] y Hillclimber [60].

3.2.4. Criterio de término de ejecución

Con respecto al criterio de término de ejecución en los trabajos revisados, la mayoría utiliza un tiempo límite fijo, que puede ir desde 1800 segundos hasta 7000 segundos, dependiendo del trabajo. Por el contrario, en [75] se usa como criterio de término, la revisión de 5×10^{10} restricciones. Y en [77] proponen una fórmula dinámica "$t_{\text{máx}} = \text{mín}(3600, \sqrt{|E|} \cdot x)$", con x un número ajustable de acuerdo al dataset.

3.2.5. Espacio de rendimiento

En cuanto al espacio de rendimiento, los trabajos [85, 86], proponen el uso del número mínimo de colores encontrado por un algoritmo en específico. En [78] proponen un espacio de rendimiento y_i definido por $y_i = \beta + 1800\alpha$, donde β corresponde a los segundos que toma converger el algoritmo a una solución que tiene una brecha de $\alpha\,\%$ desde la mejor cota inferior. En [77], se pretendía usar el algoritmo que entregue la menor cantidad de colores, pero experimentalmente notaron que en una gran cantidad de casos se dan empates. Para solucionar esto proponen una métrica de rendimiento que permita priorizar los algoritmos en base a un ranking promedio de manera que se puedan romper los empates. La fórmula está dada por $s(c, I, A) = \frac{\left|\left\{i \in I : c(i) \in B^i\right\}\right|}{|I|}$, siendo $i \in I$ una instancia del conjunto de instancias, con B^i el conjunto de algoritmos que obtiene el mejor resultado para i y $c(i)$ el algoritmo predicho para la instancia i. En [75] se propone una métrica para definir igualdad de calidad de solución. Para esto hacen introducción de un parámetro de tolerancia ϵ, de tal manera que "dos algoritmos entregaran una misma solución si los costos de ambas están a $\epsilon\,\%$ uno del otro", de esta manera un algoritmo sera "ϵ-good", si la solución obtenida no es $\epsilon\,\%$ mayor que la mejor solución en el conjunto de solvers en la instancia.

3.2.6. Modelos de machine learning

Se han realizado varios trabajos de selección automática de algoritmos usando un enfoque de *clasificación*. Como se describe en [91], éste es el enfoque clásico, en el que principalmente se busca predecir el mejor algoritmo para una instancia. Por otro lado, existen trabajos como [92], en donde lo que se busca es determinar cuáles son las características de los conjuntos de datos que se prestan para un mejor modelado mediante ciertos algoritmos de aprendizaje.

En cuanto a *regresión*, destacan [49] y [93], donde lo que se busca predecir es el tiempo de ejecución del algoritmo o la calidad de la solución.

Los trabajos reportados anteriormente sobre machine learning describen el uso de software como MATLAB, Weka, SPSS entre otros.

En [78, 75, 77], reportan que a medida que se aumenta la cantidad de características, es necesario aplicar una selección de características y reducción de dimensionalidad como Principal Component Analysis (PCA), para llevar la información a un plano bidimensional. Por otro lado, en [85, 86] para explorar el espacio de rendimiento se hace uso de Self Organization Feature Maps (SOFM) [94]. Esta técnica consiste en encontrar un mapeo desde entradas de datos de alta dimensionalidad a un espacio de características de baja dimensionalidad. De esta manera se pueden ver agrupaciones de los datos y además ayudar a obtener mejores resultados en base a la métricas utilizadas. Entre los clasificadores destacados se encuentran SVM, K-Nearest Neighbors (KNN) ya que ellos reportan los mejores resultados en los estudios.

3.3. Machine learning

A continuación se describirá el estado del arte sobre los modelos de machine learning utilizados para este trabajo.

3.3.1. Modelos de árboles

Los modelos utilizados en este trabajo corresponden a modelos de árboles basados en Boosting [95]. Éstos, por ejemplo, tienen la ventaja de ser útiles utilizando pocos datos, requieren un menor tiempo de entrenamiento, un menor conocimiento al realizar un ajuste de

parámetros y proveen de explicaciones sobre las características mas relevantes. Actualmente los modelos que dominan el estado del arte son xgboost[96], lightgbm [97] y catboost [98].

Xgboost es un modelo que destaca por ser unos de los primeros esfuerzos en reducir los tiempos de entrenamiento de modelos basados en boosting. Además de implementar regularización de hiper-parámetros sobre los árboles, es altamente escalable, permitiendo utilizar computación distribuida para procesar conjunto de datos grandes.

LightGBM, por su parte, es altamente distribuido al igual que *XGboost*. De él se destaca que fue el primer algoritmo en usar binning sobre características continuas para reducir el número de particiones de los datos. Además de implementar una nueva técnica de partición Gradient-based One-Side Sampling (GOSS), que consiste en seleccionar la partición usando todas las observaciones que tienen mayor error y una muestra aleatoria de los que tienen menor error.

Al igual que los 2 modelos anteriores, *Catboost* es altamente distribuido y tiene la particularidad de que fue diseñado para manejar de mejor manera los datos categóricos. Implementa una técnica nueva llamada Minimal Variance Sampling (MVS) que es una versión de muestreo ponderado del *Stochastic Gradient Boosting*, esto se aplica a nivel de árboles, así de esta manera las observaciones para cada árbol de boosting, son muestreadas de tal manera que maximizan el score de separación.

3.3.2. Optimización bayesiana de hiper-parámetros

Los modelos mencionados anteriormente, tienen una gran cantidad de parámetros que deben ser ajustados para obtener un buen rendimiento sobre los datos. Llevar a cabo este ajuste se puede realizar de distintas maneras: manual, búsqueda en grilla (aunque se ha reportado que este tipo de búsqueda no siempre encuentra el mejor ajuste) y búsqueda aleatoria [99]. Las estrategias mencionadas anteriormente no consideran información de los modelos entrenados previamente, lo cual podría reducir la búsqueda para enfocarse en los parámetros más prometedores. Para solucionar lo anterior se puede utilizar optimización bayesiana de hiper-parámetros, que se resume en construir un modelo de probabilidad de la función objetivo y usarlo para seleccionar los mejores hiper-parámetros para evaluarlos en la función objetivo real. De esta manera, la búsqueda de hiper-parámetros es realizada de manera informada.

Para este trabajo es de interés Sequential Model-Based Optimization (SMBO) [100] que es una formalización de optimización bayesiana, donde el modelo es entrenado varias veces de manera secuencial, y, a medida que aumentan las iteraciones, se escogen los hiper-parámetros más prometedores. Para llevar a cabo esta búsqueda, se necesita el dominio de los hiper-parámetros a buscar, una función objetivo que considere los hiper-parámetros, un modelo a ajustar, una función de selección y un historial de entrenamientos. En particular se hace uso del algoritmo de Tree Parzen Estimator (TPE) [101] que es el modelo usado para escoger los hiper-parámetros. En pocas palabras, es un modelo que aplica la regla de bayes, utilizando un umbral, para mantener 2 distribuciones: una donde se encuentre la combinación de hiper-parámetros bajo el umbral y otra sobre el umbral. De esta manera, los hiper-parámetros son muestreados evaluando ambas distribuciones sobre la función de selección.

3.4. Discusión

En la literatura revisada no se consideró el rendimiento anytime de los algoritmos, lo cual es fundamental en este trabajo. Además, los tiempos de ejecución de los trabajos anteriores son variables y pueden basarse en tiempo, cantidad de restricciones a analizar o cantidad de iteraciones para ciertos procesos. En comparación, para esta tesis, la medición del comportamiento anytime de algoritmos de distinta naturaleza, hace más adecuado utilizar restricciones temporales a los tiempos de ejecución, aunque en los trabajos revisados notan que dependiendo de la naturaleza del solver es más correcto utilizar otros criterios.

Además de las instancias y generadores propuestos en la literatura, en este trabajo se utilizará un generador basado en aprendizaje *HiSampler* [68]. Éste logra aprender cómo generar instancias difíciles para los problemas usando solvers específicos. De esta manera se pretende ampliar la variedad de los tipos de instancias.

Con respecto a la características, se propone utilizar las propuestas en los trabajos anteriores y luego realizar una selección de las más relevantes al problema. Para esto se pretende usar distintos métodos de selección de éstas a los propuesto en lo anterior, como importancia de características, *Recursive Feature Elimination (RFE)* [102] o alguna otra técnica que permita disminuir el número de ellas para evitar la alta dimensionalidad.

El espacio de rendimiento es distinto a todo lo propuesto anteriormente, a la medición

del comportamiento anytime que debe reportar, para cada instante, cual es el mejor algoritmo. Para esto, en el aspecto de clasificación se seleccionará el solver que entregue una menor cantidad de colores; en el caso de existir empates, se romperá primeramente seleccionando el solver que primero haya alcanzado la mejor solución, y en el caso de que no sea suficiente se romperá seleccionando en orden alfabético el nombre del solver. En un enfoque de regresión, se propone el uso de una métrica de distancia a la mejor solución conocida en ese instante de tiempo, de esta manera el rendimiento quedará como un porcentaje entre 0 y 1.

Finalmente, para machine learning se pretende hacer uso de modelos más cercanos al estado del arte actual como el modelo CatBoost [98], además de realizar una búsqueda de parámetros del modelo, haciendo uso de optimizadores de hiper-parámetros como hyperopt [101], y realizar una validación cruzada sobre los modelos de machine learning lo cual no es mencionado en ninguno de los trabajos anteriores.

4. SELECCIÓN AUTOMÁTICA DE ALGORITMOS ANYTIME PARA COLOREO DE GRAFOS

4.1. Selección automática de algoritmos restringida

En este trabajo se plantea usar una extensión a la metodología propuesta por Rice (Sección 2.4) denominada *Constrained Automatic Algorithm Selection*. En esta modificación se toma en cuenta cómo los recursos computacionales pueden afectar el rendimiento de los solvers. Para esto se define un nuevo conjunto R, que corresponde al espacio de recursos, en él se puede considerar el tiempo de ejecución del solver, la cantidad de memoria RAM, el número de CPUs, frecuencias de la CPU entre otras características. De esta manera, el problema queda redefinido como un mapeo de $S : F \times R \rightarrow A$, o, visto de otra manera, $\alpha = S(f(x), r)$, con r él o los recursos a considerar. Esta nueva metodología puede ser resumida en la Fig. 4.4, la cual ha sido utilizada con anterioridad en [103] que corresponde a un trabajo sobre el problema de Knapsack. Al igual que en dicho trabajo, para esta tesis se realizó una Selección Automática de Algoritmos Anytime. Además, una instancia de la metodología anterior y el marco utilizado en este trabajo puede ser visto en la Fig. 4.5.

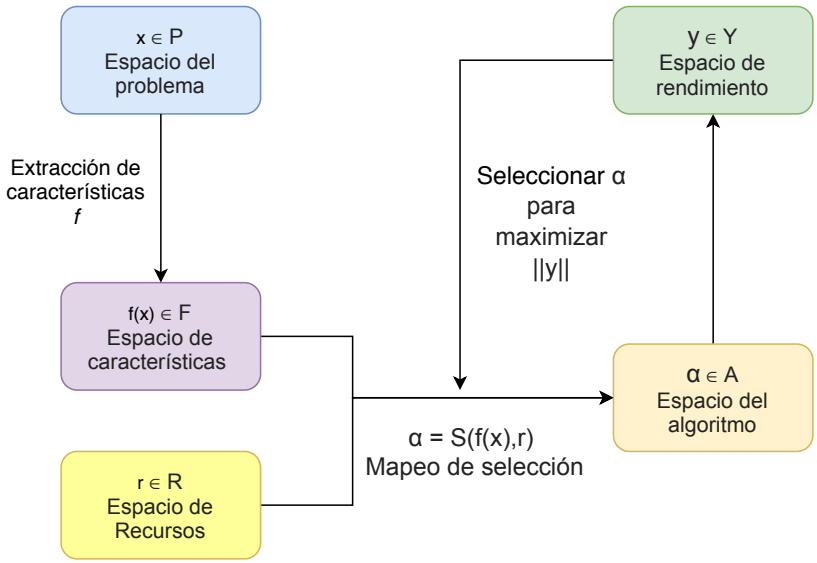


Fig. 4.4: Marco de trabajo abstracto

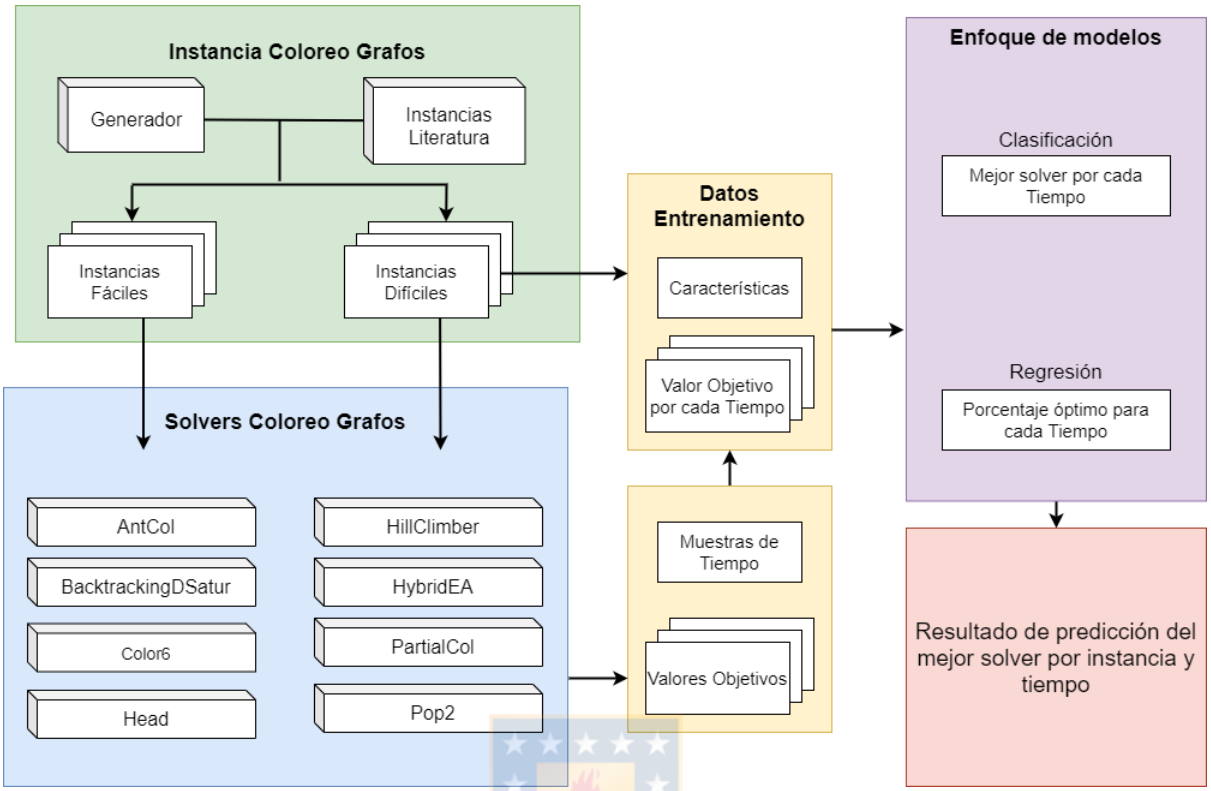


Fig. 4.5: Marco de trabajo propuesto

Los datos utilizados en la realización de este trabajo se encuentran disponibles en una carpeta virtual [9] y su descripción en el Anexo A.2. A continuación se detallan los componentes de la metodología.

4.1.1. Conjunto de solvers

Los solvers para este conjunto fueron seleccionados según su importancia en la revisión bibliográfica (ver sección 3.1.1), considerando que es de interés mantener una variabilidad en la naturaleza de los solvers, los cuales son detallados en la Tabla 4.4.

Los solvers con código fuente disponibles debieron ser modificados para seguir el enfoque anytime. Para esto, cada vez que se encuentra una nueva solución, se almacena un par *<tiempo solución, cantidad de colores>*, donde el tiempo es almacenado en milisegundos (ms). En el

[9] https://drive.google.com/drive/folders/1qJcBoPNiX3fX3lqVEjEpGBszFliEktE8?usp=sharing

caso del solver *Head*, que originalmente se ejecuta de manera paralela, se modificó el parámetro para lograr ejecuciones secuenciales, para realizar los experimentos y comparar con los demás solvers. Además fue necesario ejecutar inicialmente un solver Greedy que permita encontrar una solución inicial; para esto se uso el solver *DSATUR*. Fuera de esto, los solvers fueron ejecutados con sus parámetros por defecto.

Solver	Conseguido	Exacto/Heurístico	Tipo	Año	Lenguaje solver
AntCol	Conseguido	Heurístico	Colonia de hormigas	2008	C/C++
BacktrackingDSatur	Conseguido	Exacto	Backtracking	1979	C/C++
Color6	Conseguido	Exacto	CDCL	2014	C/C++
Head	Conseguido	Heurístico	Memetico	2018	C/C++
HillClimber	Conseguido	Heurístico	Búsqueda local	2009	C/C++
HybridEA	Conseguido	Heurístico	Híbrido Evolutivo	1999	C/C++
PartialCol	Conseguido	Heurístico	Búsqueda Local	2008	C/C++
pop2	Conseguido	Exacto	Programación Entera	2017	Python/Gurobi
PICASSO	No Conseguido	Exacto	CDCL	2019	No Disponible
GC-CDCL	No Conseguido	Exacto	CDCL	2019	No Disponible
MACOL	No Conseguido	Heurístico	Memetico	2010	No Disponible

Tabla 4.4: Resumen de los solvers encontrados en la revisión del estado del arte

Para los siguientes solvers: *PICASSO* [59], *GC-CDCL* [58], *MACOL* [62], los códigos fuente no fueron conseguidos como se mencionó anteriormente. Cabe destacar que la implementación de los algoritmos *AntCol*, *BacktrackingDsatur*, *HillClimber*, *HybridEA* y *PartialCol* fue encontrada en los códigos de apoyo del libro [13].

4.1.2. Conjunto de instancias

Para los experimentos se usaron distintos conjuntos de instancias públicas y generadores de instancias. En la tabla 4.5 se resume la cantidad de instancias por conjunto. Se recolectaron un total de $7{,}265$ instancias.

De la tabla 4.5 se puede observar que gran parte de las instancias presentes en el conjunto de datos son generadas, principalmente porque es difícil encontrar otro tipo de instancias. Con respecto a las instancias públicas, se tiene presente el conjunto más utilizado por los autores para evaluar nuevos solvers, el cual es *DIMACS*. Para facilitar el uso y homogeneizar la entrada

Dataset	Tipo	**# instancias**
Dimacs [104]	Público	136
GNP [105]	Público/Generador	340
CHI [70]	Público	1260
sports scheduling [106]	Público	10
Hugo Hernandez [74]	Público	98
BHOSLIB [67]	Público/Generador	35
Networkx Generator [66]	Generador	1361
HiSampler [68]	Generador	1025
Joe Cuberlson Generator	Generador	3000

Tabla 4.5: Cantidad de instancias por dataset

de las instancias a los solvers, todas éstas fueron transformadas a formato *DIMACS*[10].

Según [107], utilizar mayoritariamente instancias generadas es una desventaja. En el, se recomienda preferiblemente utilizar instancias basadas en el mundo real, de aquí destaca *DIMACS* y las instancias de *sports scheduling* que pertenecen a esta categoría. En segundo lugar se recomienda usar variantes de dichas instancias; sin embargo, esto no se realizó en este trabajo. En tercer lugar se sugiere utilizar instancias públicas, en este punto se tiene una gran cantidad de instancias y finalmente se usan instancias generadas, ya que ellas pueden presentar un sesgo en ciertos parámetros o esconder estructuras que puedan favorecer a ciertos algoritmos.

A continuación se detalla la metodología para generar instancias:

4.1.2.1 Generadores utilizados

Se utilizaron generadores especializados para coloreo de grafos y otros que no. Debido a lo anterior, antes de ejecutar los solvers sobre los grafos, se revisó que ciertas restricciones no sean violadas, por ejemplo, que los grafos sean conexos y no dirigidos. Por esto las instancias que eran no conexas fueron descartadas, mientras que algunos grafos poseían loops (aristas que salen y entran de un mismo nodo), eliminando estas aristas.

[10] `https://mat.gsia.cmu.edu/COLOR08/`

Según el generador de instancias se utilizaron distintos parámetros. Los generadores y sus parámetros pueden encontrarse en el anexo A.1.

Del proceso de recolección y selección de algoritmos e instancias se puede concluir que encontrar solvers con código fuente disponible e instancias no es un proceso fácil, ya que muchas publicaciones no hacen públicos sus datos y, en algunos casos, los enlaces publicados en los trabajos no se encuentran disponibles.

4.1.3. Conjunto de características

Para generar un conjunto de datos es necesario identificar qué características pueden ser relevantes para el problema a tratar y también cuáles pueden ser calculadas en tiempos razonables. Basado en los trabajos revisados se usaron las características propuestas en ellos (Sección 3.2.2), aunque algunas fueron descartadas debido al tiempo necesario para calcularlas. Tal es el caso de el vector propio de centralidad (perron-frobenius) que permite medir la importancia de un nodo dentro del grafo y la clique máximal que entrega una cota inferior del problema de coloreo de grafo. Otras como: grado de los nodos, longitud de los caminos, coeficiente de clustering global, local, pesado, beetweness centrality y excentricidad, se calcularon los estadísticos de la tabla 4.6, ya que es complicado entregar esta información sin procesar a los modelos de machine learning y se pueden obtener buenos indicadores desde ellas.

Estadísticos	**Descripción**
Máximo	Máximo valor de la lista
Mínimo	Mínimo valor de la lista
Media	Media aritmética de la lista
Mediana	Mediana de la lista
Entropía	Entropía de la lista
Variación	Medida de variación normalizada
Quintil 25	Valor que divide el 25 % más bajo de los datos
Quintil 75	Valor que divide el 75 % más bajo de los datos

Tabla 4.6: Estadísticos calculados

Algunas de las características fueron ejecutadas usando C++ con la librería boost [108],

mientras que otras se calcularon usando python con graph_tool [109] y networkx, según disponibilidad de las características deseadas.

Del análisis realizado luego de calcular todas las características se encontró que existian 313 instancias repetidas, provenientes del generador networkx en las clases *ws, cws, nws, dense, gnm*, finalmente se utilizaron 6380 instancias.

4.1.4. Configuración experimental

Los experimentos se ejecutaron sobre un clúster con el administrador de trabajo slurm [110]. Cada nodo consta de un procesador Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80GHz con 64 GB de memoria RAM, con la distribución Ubuntu 18.04.2 LTS. Para cada instancia se definió un tiempo máximo de ejecución de 2 horas. Sin embargo, para acortar el tiempo de ejecución total, debido a la cantidad de instancias, se definió un tiempo $t = 300000$ ms (5 minutos) de corte en el caso de que no se logre mejorar la solución en dicha ventana de tiempo. En consecuencia, si durante alguna ejecución, un algoritmo pasaba más de 5 minutos sin mejorar su mejor solución vigente, éste era detenido. Si bien la naturaleza de algunos solvers es probabilística, los experimentos se ejecutaron una sola vez por temas de tiempo.

4.2. Análisis de la ejecución de los solvers

Con respecto a la ejecución es importante destacar que los solvers *Pop2* y *Head* no encuentran soluciones para todas las instancias, esto sucede principalmente porque *Pop2* requiere mucha memoria RAM en algunas instancias y en otras excede el tiempo de corte para encontrar una solución. *Head* por su parte solo excede el tiempo para encontrar solución, ya que en cada iteración se ejecuta dos veces el algoritmo *Tabucol*, cada una con distintos parámetros; *Head* es un algoritmo pensado para ser ejecutado usando paralelismo, por lo cual en este estudio corre en desventaja. *Pop2* tuvo 1676 instancias donde no se reportan soluciones, mientras que Head no reporta solución en 182 instancias.

Para generar un conjunto de datos que represente el comportamiento de los solvers en función del tiempo es importante analizar los resultados obtenidos en la ejecución de éstos. En la Fig. 4.6 se muestran todos los instantes de tiempo en los que los solvers reportan nuevas

soluciones. En ella se muestra una línea negra punteada que indica el instante de tiempo en el que dejan de existir cambios en el solver ganador (la intersección entre la línea negra punteada y la curva), que es, el tiempo donde ya se encontró la cantidad mínima de colores para todas las instancias. Luego de la intersección, los solvers *color6* y *pop2* siguen reportando soluciones pero sin mejorar la solución actual.

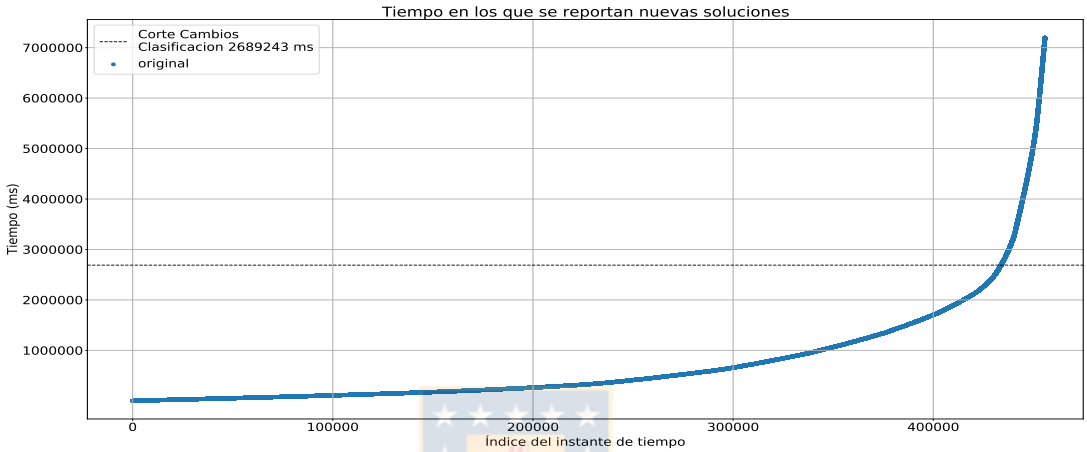


Fig. 4.6: Tiempos en los que los solvers reportan nuevas soluciones para todas las instancias.

Como se explicó anteriormente, la Fig. 4.6 representa el comportamiento donde se encuentran soluciones. Sin embargo, si se quisiera conocer dónde se concentra la mayor cantidad de ellas, se debe observar la Fig. 4.7. Esta figura muestra que la mayoría de las soluciones son reportadas en los primeros tiempos de ejecución. Notando que la figura corresponde a los primeros dos segundos de ejecución, es posible concluir que gran parte de la variabilidad se encuentra dentro de los primeros instantes.

Fig. 4.7: Soluciones reportadas por instante de tiempo

Para analizar lo anterior de mejor manera se propuso clasificar las instancias utilizadas en base a los tiempo de ejecución y según las soluciones obtenidas, para ello se definieron 3 clases:

1. **Trivial:** Los solvers greedy entregan la misma solución que color6 y termina su ejecución en un tiempo menor a 10 segundos.
2. **Easy:** Si el 50 % de los solvers encuentran una mejor solución que greedy dentro de los primero 10 segundos.
3. **Hard:** Si no pertenece a ninguna de las anteriores.

En la Fig. 4.8 se muestra la distribución de las clases presentes en el conjunto de instancias, donde predominan *Easy* y *Hard*.

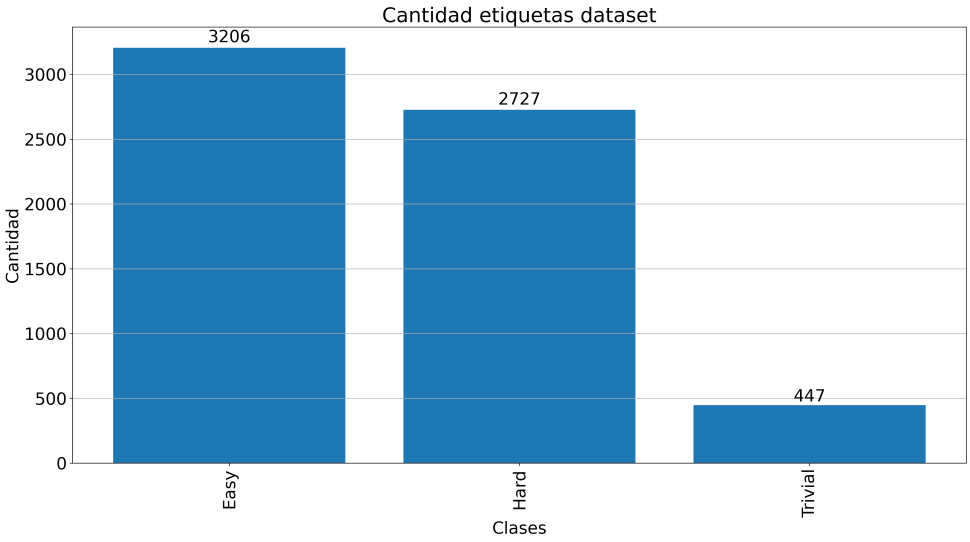


Fig. 4.8: Clasificación instancias

Basado en el resultado de la Fig. 4.8, aproximadamente el 50 % de las instancias pertenece a las instancias *Easy*, en segundo lugar un aproximado de 40 % a las instancias *Hard* y una pequeña parte pertenece a las *Trivial*. De este análisis se concluye que la mayor densidad de soluciones se encuentran en los primeros instantes. A medida que avanza el tiempo de ejecución se entregan cada vez menos soluciones y con una mayor diferencia de tiempo entre ellas.

Para generar los tiempos a utilizar, se decidió usar una porción de los tiempos que se obtienen en la ejecución, ya que es complejo manejar la cantidad de datos si se consideran todos los instantes de tiempos donde se reportan soluciones, además de que existen instantes donde no hay variación de solver ganador, por lo que no aportan mucha información. Para evaluar si tiene relevancia considerar los primeros instantes de ejecución, se generaron dos curvas que simulan el comportamiento de los algoritmos a lo largo del tiempo: una denominada *lineal* porque durante los primeros instantes de ejecución tiene una separación lineal que luego sigue un comportamiento exponencial y la otra *exponencial*, ya que sigue un comportamiento exponencial en toda ella. Estas curvas pueden ser vistas en la Fig. 4.9.

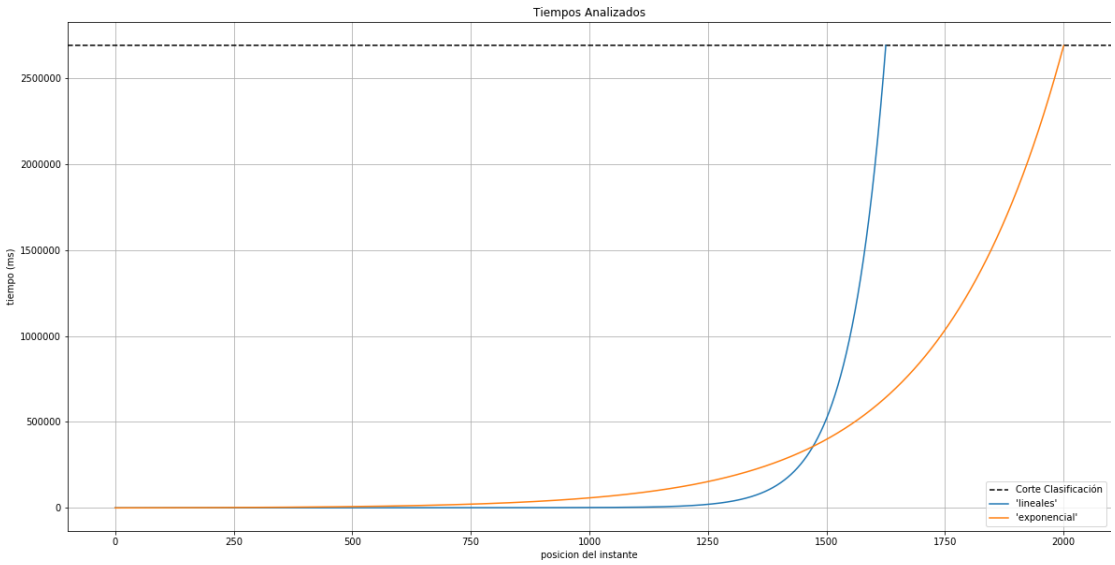


Fig. 4.9: Tiempos generados para analizar

Usando los tiempos de la curva *lineal* en la Fig. 4.10 se observa la distribución de los tiempos en los cuales se reporta la obtención de la mejor solución para una instancia en específico. Para representar mejor esto, los tiempos se redujeron a *bines* de diferentes cantidades de tiempo, lo cual es representado por la dimensión del color en cada *bin*. De aquí es posible ver que la gran mayoría de las instancias obtienen sus mejores soluciones en los primeros ms. Se debe considerar que a medida que se avanza en los bines, la diferencias de tiempo entre ellos es mayor.

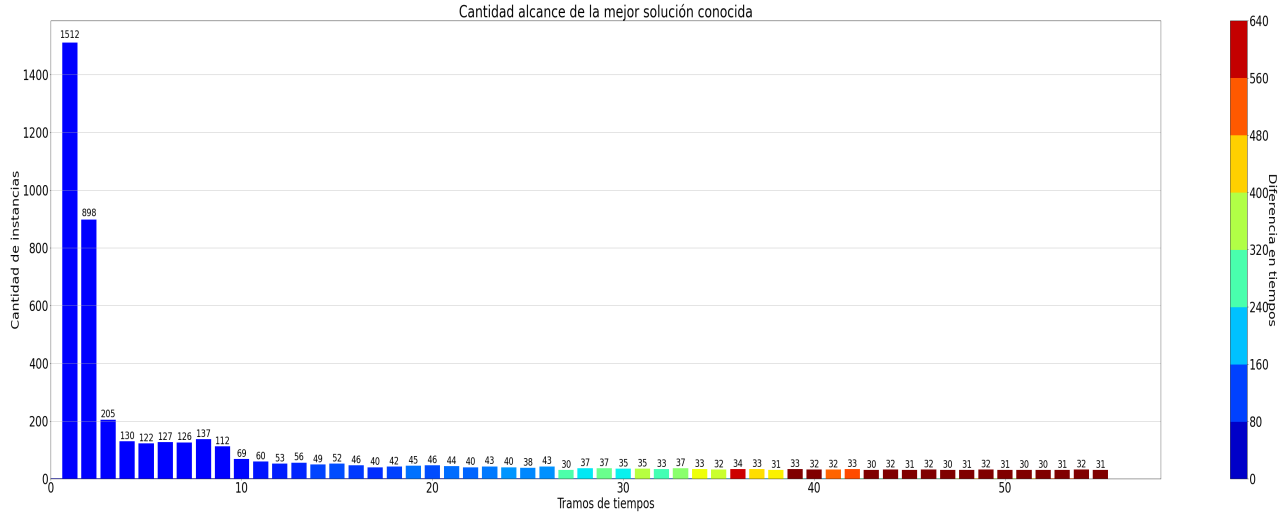


Fig. 4.10: Tiempo en que una instancia alcanza el mejor valor objetivo

Una vez obtenidos los resultados, es necesario definir un criterio de evaluación que se ajuste a los fines de este trabajo, para esto se considera utilizar la Cercanía a la Mejor Solución Encontrada (CMSE), que es una medida de distancia de la solución actual de un solver en particular con respecto a la mejor solución encontrada para esa instancia. Para este trabajo se considera como mejor solución la menor cantidad de colores a utilizar. Sea un solver $s \in A$, con A el conjunto de todos los solver, una instancia del problema $i \in P$, con P el conjunto de todos los problemas y un tiempo $t \in T$, con T los tiempos generados para cada conjunto (curva lineal y exponencial). Sea Δ_i el grado máximo de los nodos de una instancia $i+1$ y Δ el máximo de los $\Delta_i \in P$. Sea γ_i la mejor solución encontrada por los algoritmos greedy para la instancia i y sea α_i la mejor solución encontrada por todos los algoritmos para la instancia i, y x_t corresponde a la solución actual en el instante de tiempo t. Con lo anterior la CMSE queda definida como:

$$CMSE(x_t, \Delta, \Delta_i, \gamma_i, \alpha_i) = \begin{cases} 0 & x_t \leq \Delta \\ \frac{\Delta - x_t}{\Delta - \Delta_i} \times 0{,}4 & \Delta_i < x_t \leq \Delta \\ (1 - \frac{x_t - \gamma_i}{\Delta_i - \gamma_i}) \times 0{,}4 + 0{,}4 & \gamma_i < x_t \leq \Delta_i \\ (1 - \frac{x_t - \alpha_i}{\gamma_i - \alpha_i}) \times 0{,}2 + 0{,}8 & \alpha_i < x_t \leq \gamma_i \end{cases}$$

En el caso de que $t \notin T$, se busca el tiempo $t_T \in T$ más cercano a t, para esto se comparará $\forall t_T \in T : min(|t_T - t|)$ y de esta manera se utiliza el tiempo obtenido de la función anterior en reemplazo de t; en caso contrario de que $t \in T$ se utiliza t.

El comportamiento anytime de los solvers puede ser visto en la Fig. 4.11, en el eje *y* se muestra el CMSE con respecto a la mejor solución conocida para la instancia, que es calculada con respecto a todos los solvers. En la figura se observa el caso de la instancia número 1509, en la que, a medida que avanza el tiempo, la mayoría de los solver mejoran sus soluciones y consultando en distintos instantes de tiempos es posible obtener distintos solvers que obtengan la mejor solución.

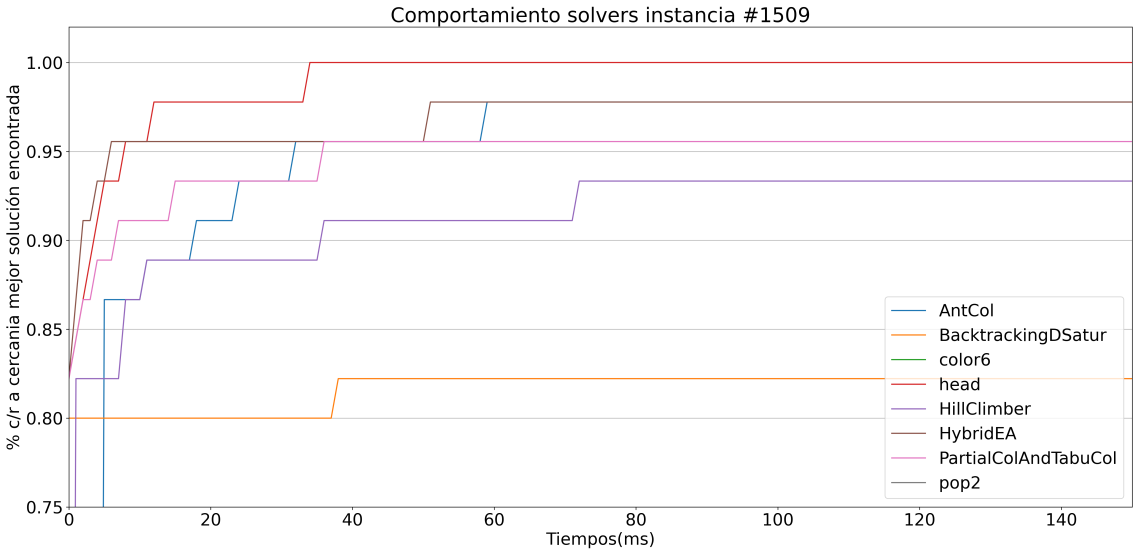


Fig. 4.11: Evolución soluciones de los solvers para una instancia específica

En la Fig. 4.12 en el eje x están las instancias, en el eje y está el tiempo, y los colores presentes en la figura muestran el solver ganador para cada instancia y tiempo específico. Este solver ganador es el que tiene la mejor solución en el instante de tiempo. En la figura se muestra la variación del solver ganador para un subconjunto de 3107 instancias. Al igual que en el caso anterior, se observa que en distintos tiempos existen distintos solvers ganadores. De esta figura se puede concluir que existe una tendencia en el solver ganador, en este caso *Head*, y que, en muchas instancias, el solver ganador no varía durante su ejecución. Esto puede ser porque el solver que obtiene la mejor solución se mejora a sí mismo y sigue adquiriendo la mejor solución o ya no se obtiene mejores soluciones independientemente del tiempo de ejecución. Es importante destacar que en la figura no se está considerando los empates que pueden suceder durante la ejecución de los algoritmos.

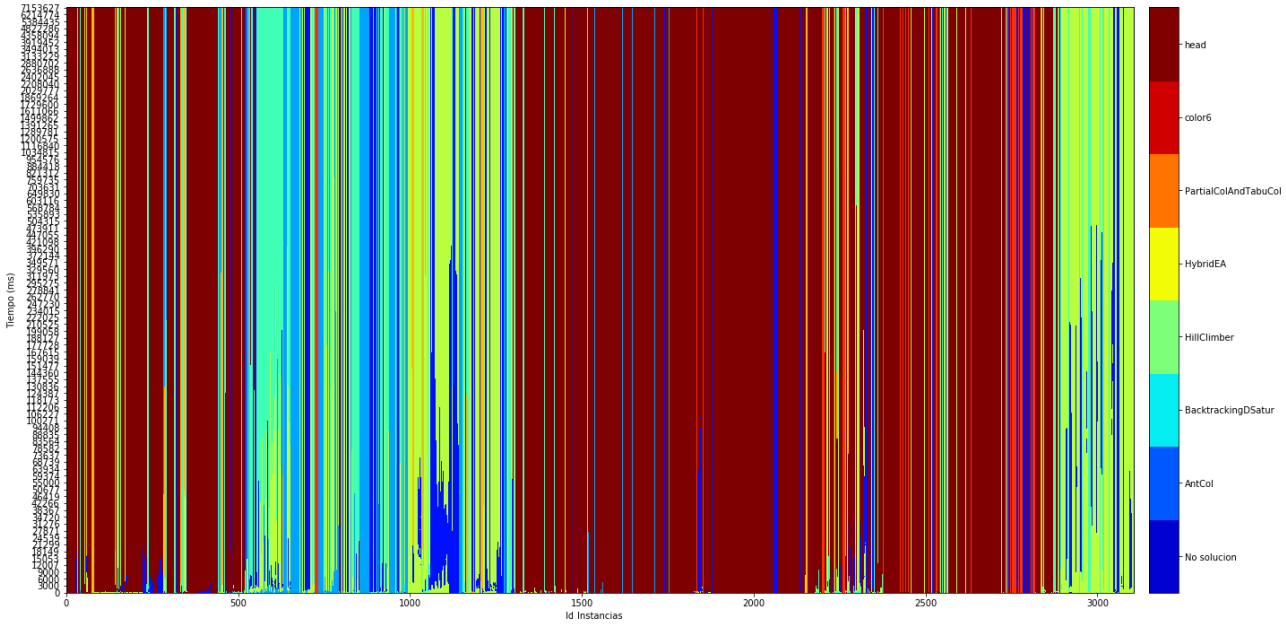


Fig. 4.12: Algoritmo ganador para distintas instancias en distintos instantes de tiempo

Tal como se dijo anteriormente, en la Fig. 4.12, no se puede apreciar si entre los solvers se generan empates; para ello se generó la Fig. 4.13, que representa la proporción de veces que el algoritmo adquiere la mejor solución conocida por el conjunto de solvers en un instante de tiempo determinado para todas las instancias. Para esto se agrupó el tiempo en *bines*, tal como se realizó en la Fig. 4.10. Para el cálculo se usaron los tiempos *lineales* y de manera similar se calculó para los tiempos *exponenciales* en la Fig. 4.14. En ambos casos se puede apreciar la predominancia del solver *Head* pasado los primeros segundos de ejecución, aunque en el caso de *exponencial* se puede notar que *Head* obtiene la predominancia antes. Con el fin de realizar un análisis más profundo se decidió hacer el ejercicio de remover el solver *Head* del conjunto de solvers, obteniendo la Fig. 4.15, de aquí se puede observar que *HybridEA* predomina sobre los demás solvers. De la misma manera, se eliminó el solver *Head* y *HybridEA* en la Fig. 4.16 y es posible observar, un escenario de mayor competencia, donde el solver *PartialCol* y *AntCol* obtienen las mejores soluciones. En esta figura *pop2* tiene un poco más de relevancia en los instantes finales de ejecución.

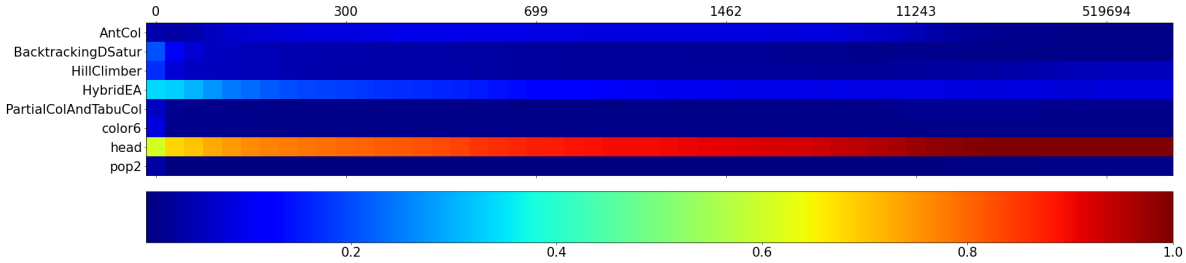


Fig. 4.13: Predominancia promedio de solver ganador usando curva de tiempo lineal y todos los solvers

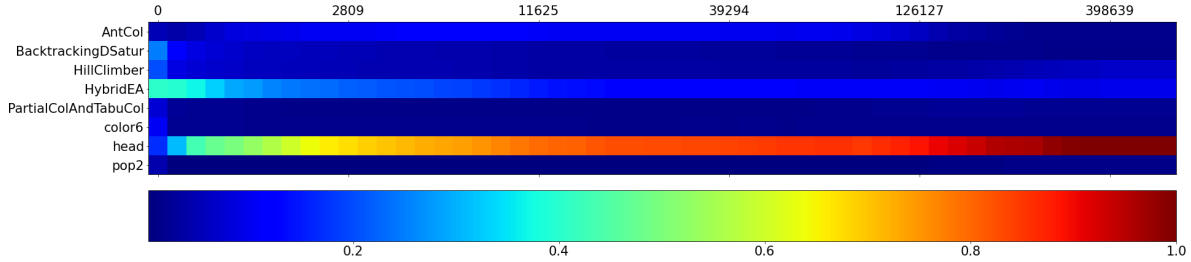


Fig. 4.14: Predominancia promedio de solver ganador usando curva de tiempo exponencial y todos los solvers

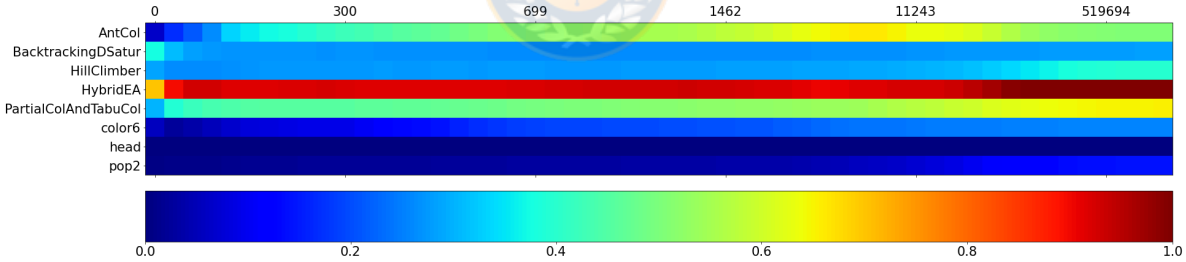


Fig. 4.15: Predominancia promedio de solver ganador usando curva de tiempo lineal y sin Head

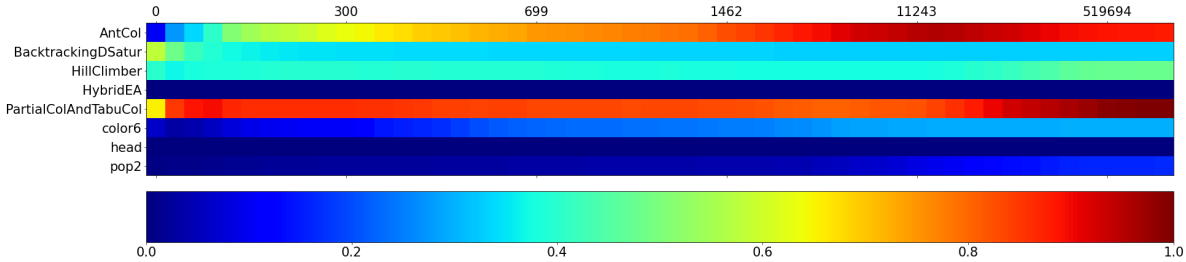


Fig. 4.16: Predominancia promedio de solver ganador usando curva de tiempo lineal y sin Head ni HybridEA

De las Fig. 4.13 y Fig.4.14 se puede concluir que es más interesante analizar la curva *lineal* que la curva *exponencial*, ya que existe una mayor competencia en la primera que en la segunda. Además de los casos analizados en las imágenes anteriores, pareciera que sin importar la combinación de solvers a utilizar, siempre existe un algoritmo que domina, aunque se tienen diferentes proporciones.

4.3. Enfoque

Para lograr tener un oráculo predictor del solver ganador, basado en los tiempos de ejecución, se optó por seguir un enfoque de aprendizaje supervisado. Para esto es necesario crear un conjunto de datos que represente la situación y sobre el cual se puedan entrenar los modelos. El objetivo es predecir el solver que entregue el mejor rendimiento para una instancia y tiempo dados. Para esto se plantea enfoques de clasificación y regresión.

4.3.1. Clasificación

Aquí el objetivo es entregar el mejor solver ganador, sin importar la consideración de empates. Para esto, cada par $<$*solución,tiempo*$>$ entregado por lo solvers es considerado. Desde el enfoque de solución seguido por este trabajo, primero se deben homogeneizar los tiempos de las soluciones, para lo cual se deben mapear los tiempos obtenidos por la ejecución del solver a los tiempos a analizar. Para esto, es relevante contar con una función de distancia entre un tiempo y otro, considerando que el tiempo de solución debe ser mayor o igual al tiempo analizado; ésto es importante, ya que, cuando no se considera la restricción de que el tiempo de solución debe ser mayor o igual, se obtienen peores resultados. Una vez ajustados los tiempos, corresponde que a cada uno de ellos, se le seleccione el solver que entrega una menor cantidad de colores para que sea considerado el mejor solver en ese instante de tiempo.

De este proceso y analizando varios conjuntos de solvers a utilizar, se observa en la tabla 4.7 la proporción de ellos presente en el conjunto. De aquí se concluye lo mismo que en la Fig. 4.16; a medida que se quitan solvers, las proporciones de todos los solvers varían y es fácil notar que en estas combinaciones siempre hay uno que destaca sobre los demás, haciendo que las proporciones de las etiquetas se encuentren desbalanceadas. También es importante notar que, a pesar de que existen solvers que tienen un bajo porcentaje de presencia en los

conjuntos, éstos no se eliminan porque se pretende analizar qué es lo que sucede cuando todos los solvers compiten.

	Conjuntos de Solvers			
Solver	**Lineal**	**Lineal S/head**	**Lineal S/head S/HybridHEA**	**Exponencial**
Sin Solución	0.000649	0.000670	0.000671	0.000057
AntCol	0.046406	0.127403	0.306623	0.021379
BacktrackingDsatur	0.022057	0.104216	0.133904	0.010482
HillClimber	0.024238	0.081851	0.116173	0.018106
HybridHea	0.089153	0.640234	0.000000	0.047019
PartialCol	0.000220	0.034952	0.431544	0.000426
Color6	0.009843	0.010176	0.010423	0.008842
Head	0.807039	0.000000	0.000000	0.892589
pop2	0.000396	0.000498	0.000662	0.001100

Tabla 4.7: Proporción de presencia de solvers en diferentes conjuntos de solvers

Como puede ser visto en la tabla 4.7, existe un desbalance en la proporción del algoritmo ganador generalizado, por lo cual es conveniente evaluar el uso de pesos para combatir este efecto. De la misma manera, se intentó generar datos sintéticos utilizando como base los datos existentes, pero se encontró la complejidad de que para generar estos datos se debe hacer que pertenezcan a una instancia en específico, por lo cual fue descartado.

4.3.2. Regresión

En este enfoque se trata de representar el CMSE, con respecto a la mejor solución conocida por todos los solvers en un determinado instante de tiempo. Para esto, se consideran ciertos valores como puntos de decisión, como: la cota superior teórica del problema (máximo grado de los nodos más uno), la mejor solución greedy entre RLF y DSATUR, la peor y la mejor solución encontrada por los solvers. Primeramente se intentó hacer una función lineal entre la cota superior y la mejor solución encontrada, pero esto lleva a distribuciones donde la mayoría de los datos se encuentran concentrados alrededor del 0 y del 1. Considerando lo anterior se intentó utilizar distintos puntos de control, para asignar CMSE por tramos y de esta manera balancear mejor la distribución de los porcentajes, como se puede ver en la tabla 4.8. De la

misma manera que en clasificación se tuvo que realizar un ajuste a los tiempos entregados por las soluciones.

Lineal	AntCol	Backtracking Dsatur	Color6	Head	HillClimber	HybridEa	PartialCol	pop2
0 - 0.1	0.074681	0.003675	0.117242	0.051195	0.055988	0.013877	0.036886	0.735886
0.1 - 0.2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.2 - 0.3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.3 - 0.4	0.000000	0.000000	0.448893	0.000000	0.000000	0.000000	0.000000	0.000000
0.4 - 0.5	0.001003	0.000782	0.056693	0.000000	0.001254	0.001243	0.001243	0.000000
0.5 - 0.6	0.000013	0.001554	0.048785	0.000607	0.000007	0.000373	0.000272	0.000027
0.6 - 0.7	0.001266	0.004963	0.028959	0.001686	0.000028	0.001374	0.001076	0.000142
0.7 - 0.8	0.117816	0.409999	0.029197	0.042707	0.094289	0.027134	0.077519	0.184454
0.8 - 0.9	0.273622	0.425513	0.155018	0.065075	0.507530	0.215348	0.305766	0.057841
0.9 - 1	0.531599	0.153513	0.115212	0.838730	0.340903	0.740651	0.577238	0.021650

Tabla 4.8: Proporción de valores Lineal

Como uno de los objetivos es realizar una comparación entre el modelo de regresión y el modelo de clasificación para determinar cuál es el más adecuado para resolver esta tarea, se propone entrenar un modelo de regresión por cada solver y luego agrupar todos los modelos de regresión en uno solo, para comparar los CMSE y entregar el índice del más alto, tal como se muestra en la Fig. 4.17. También es relevante considerar que para comparar ambos modelos se utilizan las mismas características.

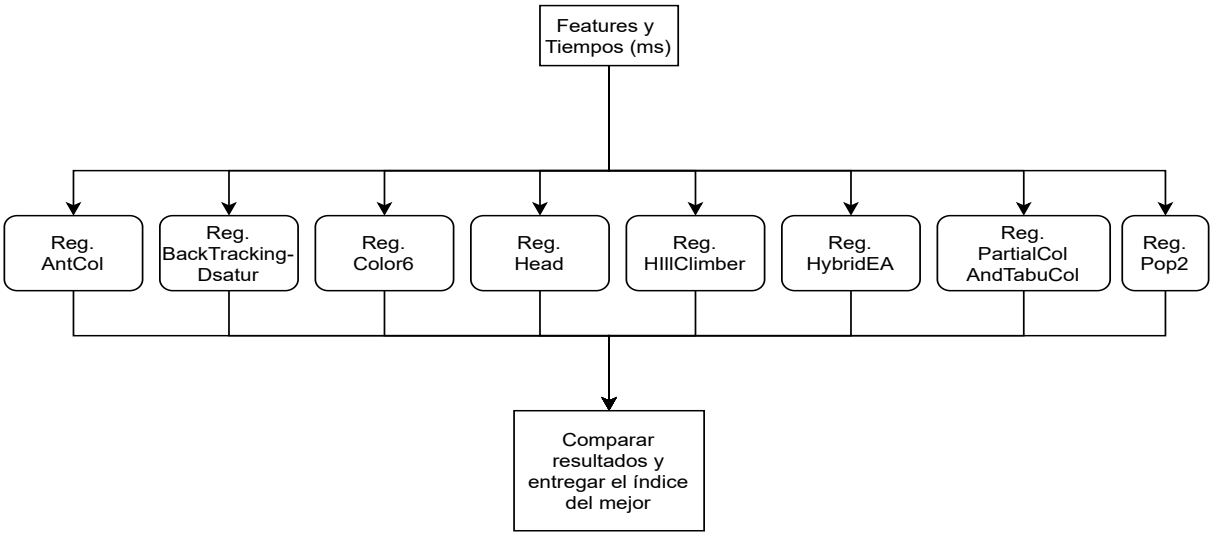


Fig. 4.17: Modelo regresor

Con respecto al entrenamiento de ambos enfoques, se realizaron pruebas utilizando validación cruzada [111]. Para esto fue necesario dividir el conjunto en dos subconjuntos: uno de entrenamiento y uno de prueba. Al realizar Cross validation (CV) el conjunto de entrenamiento se dividió en dos, uno de entrenamiento y otro de validación. Para todas las evaluaciones el conjunto de prueba se mantuvo fijo, es decir, se utilizaron las mismas instancias mientras que entrenamiento y validación fueron separados siguiendo un enfoque de *stratify-group-k-folds* (ver sección 2.3). Como la separación se hace tanto por grupo y clase, se requiere que ambos valores sean enteros. Para aplicar esta técnica al modelo de regresión, se agruparon los valores de CMSE en rangos de 10 %, solo para realizar el muestreo estratificado. Es relevante destacar la separación en grupos, ya que, de lo contrario, las instancias no se encuentran totalmente contenidas en un conjunto o en otro. Los modelos pueden obtener buenos resultados sobre el conjunto de validación pero en la evaluación se obtienen peores resultados.

Finalmente, el flujo de trabajo sobre los que se ejecutaran los algoritmos puede ser visto en la Fig. 4.18. Existen dos tiempos: un tiempo de selección, que corresponde al cálculo de las características y a la predicción del modelo, y otro tiempo de resolución que corresponde a la ejecución del solver seleccionado. De esta manera uno de los objetivos es reducir lo más posible el tiempo de selección sin tener que sacrificar el rendimiento del modelo.

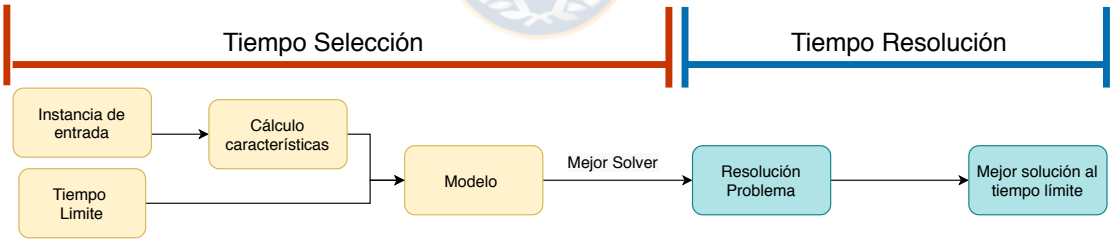


Fig. 4.18: Flujo de los modelos

4.4. Características utilizadas

Para la estandarización de los datos se utilizó un escalador robusto de *sklearn* [112]. Debido a que luego del análisis exploratorio de las características, se encontró que existen instancias con valores extremos, los cuales afectan a la estandarización, y, como no existe intención de eliminarlas del conjunto, es más adecuado estandarizar usando el primer y tercer cuartil. Lo anteriormente mencionado se descubrió luego de utilizar diferentes escaladores como *StandarScaler* y *Min-maxScaler*, pero debido a lo valores extremos no se obtuvieron buenos

resultados.

Se recolectaron 81 características y experimentalmente se fueron reduciendo el número de las utilizadas, para evitar el sobreajuste de los modelos, por lo tanto se buscaron métodos para hacer una selección de características y determinar las más relevantes.

Primero se realizó una limpieza de datos, para esto se eliminaron columnas que contienen Not A Number (NAN) (5 características) y las características que tienen poca varianza (11 características), ya que se asume que éstas entregan poca información. Luego, se intentó evaluar las características basadas en un modelo. Para evitar el efecto de desbalanceo de las clases se uso un BalancedRandomForest [113] de la librería *imbalanced learn* [114]. Con él se determinó que 14 características no aportaban mayor información, como se ve en la Fig. 4.19. En esta primera iteración se dejaron 61 características relevantes al problema.

De las 61 características, se realizó un análisis de correlación donde se obtuvo que 42 características están altamente correlacionadas entre sí, las cuales son candidatas a ser eliminadas. De aquí es necesario realizar una segunda iteración, donde se ingresaron las 61 características para evaluar cada una de ellas en distintos escenarios y criterios; entre ellos se utilizaron la correlación de Pearson, Chi^2, Regresión logística, RFE usando Regresión logística, Random Forest [35] y LightGBM [97]. Luego, se seleccionan aquellas características, donde al menos 4 de los métodos mencionados hayan seleccionado la característica. De este proceso quedaron 36 características consideradas como las más útiles.

La idea detrás de esto, es usar tres tipos de métodos de selección de características como lo son filtro, envoltura y embebido; y hacerlos votar sobre qué características son relevantes. Luego, con las 36 características se realizaron pruebas con cinco modelos distintos: Balanced Random Forest (brf), Balanced Bagging Classifier(bcc) [115], LightGBM (lgm), Catboost (cat) [98] y XgBoost (xgb), [96], a estos últimos 3 modelos se le entregaron pesos de clases, para ayudar a combatir el desbalance de éstas.

En un inicio los modelos se evaluaron utilizando la métrica MCC ya que según lo reportado en [41], esta métrica era adecuada para ser utilizada con desbalance de clases, pero al evaluar los resultados se obtuvo que la métrica favorecía a la clase con mayor presencia en el conjunto, como se puede ver en la anexo A.3. Así se puede concluir que maximizar el MCC causa que el modelo se enfoque en predecir de mejor manera la clase más predominante en el dataset, dejando de lado la predicción de las demás clases. Es por esto que se realizo una nueva iteración

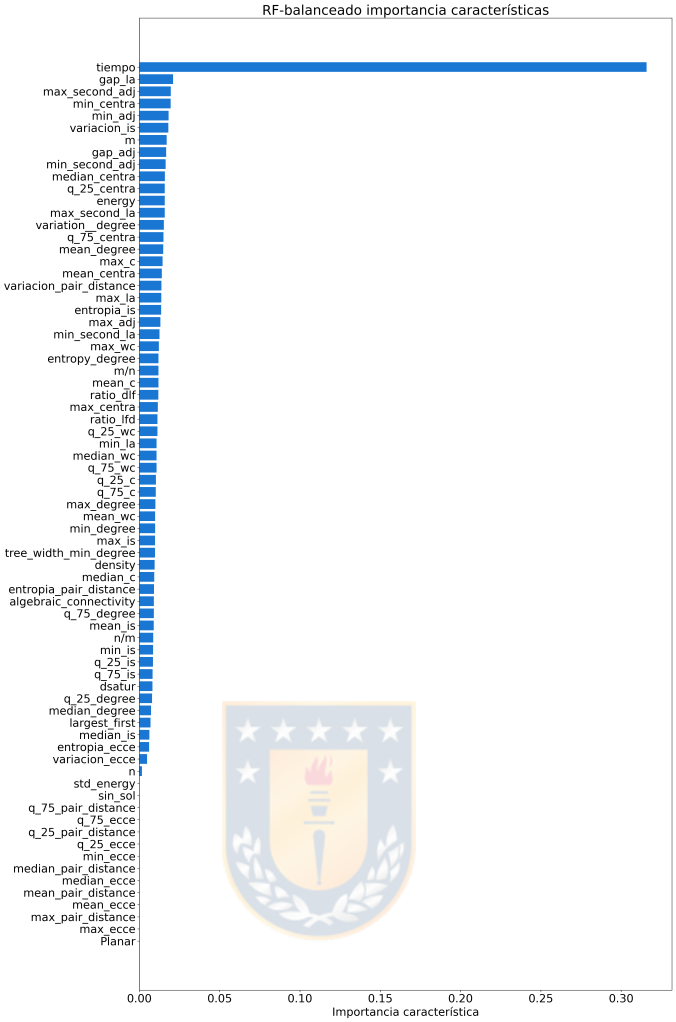


Fig. 4.19: Importancia Características Random Forest Balanceado

en el entrenamiento para optimizar la *media geométrica*, ya que, al realizar una nueva búsqueda sobre bibliografía se encontró este trabajo [116], donde se menciona que MCC no es una métrica adecuada cuando existe un desbalanceo de clases.

Los resultados pueden ser vistos en la tabla 4.9. De aquí el mejor modelo obtenido es Catboost, por los resultados obtenidos en Geometric Mean (GM); sin embargo, de estos resultados se puede ver que en esta métrica existe un alto nivel de variación con respecto a los valores en la media. Esto sucede por el desbalanceo de clases que existe en el dataset.

Métrica	Balanced RF	Balanced BC	Catboost	XgBoost	LightGbm
ACCURACY	0.9 ± 0.01	0.88 ± 0.01	0.89 ± 0.02	0.92 ± 0.01	0.94 ± 0.0
RECALL_W	0.9 ± 0.01	0.88 ± 0.01	0.89 ± 0.02	0.92 ± 0.01	0.94 ± 0.0
F1_W	0.91 ± 0.01	0.89 ± 0.01	0.9 ± 0.02	0.92 ± 0.01	0.94 ± 0.0
PRECISION_W	0.92 ± 0.0	0.91 ± 0.01	0.92 ± 0.01	0.93 ± 0.0	0.94 ± 0.0
COHEN	0.74 ± 0.01	0.7 ± 0.02	0.71 ± 0.05	0.78 ± 0.01	0.84 ± 0.01
MCC	0.75 ± 0.01	0.71 ± 0.02	0.72 ± 0.04	0.78 ± 0.02	0.84 ± 0.01
GM	0.39 ± 0.19	0.36 ± 0.11	0.62 ± 0.22	0.32 ± 0.12	0.3 ± 0.09
GM_MACRO	0.85 ± 0.04	0.84 ± 0.04	0.89 ± 0.03	0.82 ± 0.04	0.81 ± 0.03
GM_MICRO	0.94 ± 0.0	0.93 ± 0.01	0.93 ± 0.01	0.95 ± 0.0	0.97 ± 0.0
GM_W	0.94 ± 0.0	0.92 ± 0.01	0.93 ± 0.01	0.94 ± 0.01	0.94 ± 0.01
F1_MACRO	0.61 ± 0.04	0.57 ± 0.04	0.58 ± 0.05	0.61 ± 0.05	0.68 ± 0.05

Tabla 4.9: Comparación modelos base usando 36 características 10-CV

Desde el punto de vista de la importancia de características, destaca que en todos los modelos la característica más importante fue el tiempo, y que las subsiguientes están relacionadas con el grado de los nodos, el coeficiente de clustering, la variación de la distancia entre pares de vértices y valores espectrales de los grafos. Si bien en [77] se reporto el uso de estadísticos sobre características que corresponden a vectores de datos, del análisis realizado se puede concluir que muchos de los estadísticos no son relevantes por estar altamente correlacionados entre sí.

En una tercera iteración se entreno Catboost con las 36 características de los resultados anteriores, a este modelo se le añadió un detector de sobreajuste, y se reviso la importancia de las características; donde solo 20 tienen una mayor relevancia. De la misma manera, para medir la variación del rendimiento al modificar las características, experimentalmente se realizaron pruebas seleccionando las 6 características más relevantes del modelo. De la misma manera que en la tabla 4.9, en la tabla 4.10, se muestran los resultandos usando 6 características, donde la tendencia es la misma que es la tabla 4.9, solo que se obtiene valores mas bajos en las métricas.

Métrica	Balanced RF	Balanced BC	Catboost	XgBoost	LightGbm
ACCURACY	0.89 ± 0.01	0.88 ± 0.01	0.86 ± 0.02	0.89 ± 0.01	0.93 ± 0.01
RECALL_W	0.89 ± 0.01	0.88 ± 0.01	0.86 ± 0.02	0.89 ± 0.01	0.93 ± 0.01
F1_W	0.9 ± 0.01	0.89 ± 0.01	0.87 ± 0.02	0.9 ± 0.01	0.93 ± 0.01
PRECISION_W	0.92 ± 0.01	0.91 ± 0.01	0.9 ± 0.01	0.92 ± 0.01	0.93 ± 0.01
COHEN	0.71 ± 0.02	0.7 ± 0.02	0.65 ± 0.04	0.73 ± 0.02	0.79 ± 0.02
MCC	0.72 ± 0.02	0.71 ± 0.02	0.66 ± 0.03	0.73 ± 0.01	0.79 ± 0.02
GM	0.3 ± 0.09	0.32 ± 0.09	0.57 ± 0.2	0.27 ± 0.09	0.2 ± 0.08
GM_MACRO	0.81 ± 0.03	0.82 ± 0.04	0.87 ± 0.03	0.8 ± 0.03	0.76 ± 0.03
GM_MICRO	0.94 ± 0.01	0.93 ± 0.01	0.92 ± 0.01	0.94 ± 0.0	0.96 ± 0.0
GM_W	0.93 ± 0.01	0.93 ± 0.01	0.91 ± 0.01	0.93 ± 0.01	0.92 ± 0.01
F1_MACRO	0.55 ± 0.03	0.53 ± 0.03	0.52 ± 0.03	0.54 ± 0.04	0.6 ± 0.05

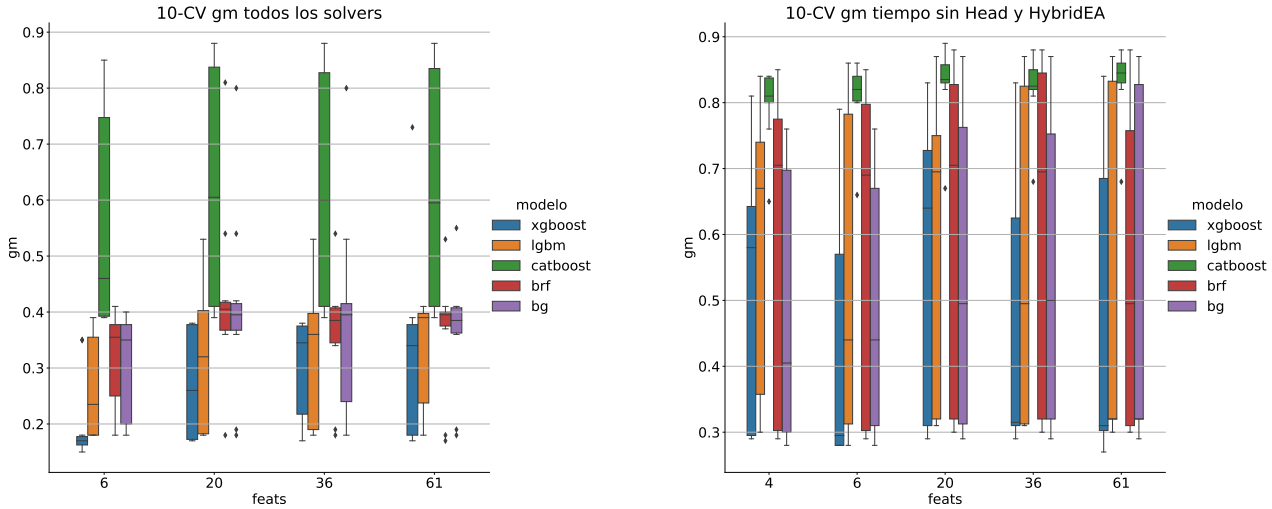
Tabla 4.10: Comparación modelos base usando 6 características 10-CV

Como es importante seleccionar la menor cantidad de características para ahorrar tiempo en el cálculo de ellas, es que se realizo un análisis de cómo varía el rendimiento de los modelos según distintas cantidades de características. Para esto se utilizó como medida de comparación la métrica GM. Además se realizó una validación cruzada usando 10 folds solo con el conjunto de entrenamiento y con peso para las clases, para ayudar a combatir el desbalance. También se agregó el detector de sobreajuste en los modelos que tuvieran esta opción disponible.

Para esto se analiza la curva de tiempo *lineal* usando todos los solver (Fig. 4.20a) y quitando los solvers *Head* y *HybridEA* (Fig. 4.20b) para clasificación. De lo anterior, se observa que en el segundo caso se obtienen mejores resultados y sin mayores variaciones en contraste a la primera situación. Esto se debe principalmente a la distribuciones de los solvers en los conjuntos, ya que tal como se puede ver en la Tabla 4.7, el primer caso se encuentra desbalanceado.

Se observa que a medida que se utiliza una mayor cantidad de características, se obtienen mejores valores en GM. Aunque las mejoras no son tan significativas en cuanto a la métrica, se observa que con 36 características y en particular Catboost obtiene los mejores valores en ambos casos. Se puede concluir que el uso de 6 características parece ser el más adecuado, ya que se obtiene un rendimiento comparable con el uso de 36 características y permite ahorrar 569 ms en promedio (calculado sobre el conjunto de test), lo cual puede ser significativo para

competir en los primeros instantes de ejecución de los solvers. Los resultados obtenidos verifican que el tiempo es la característica más importante y que a medida que se agregan características no logran impactar significativamente en la calidad del modelo.



(a) GM Todos los solvers 10-CV variando cantidad de características

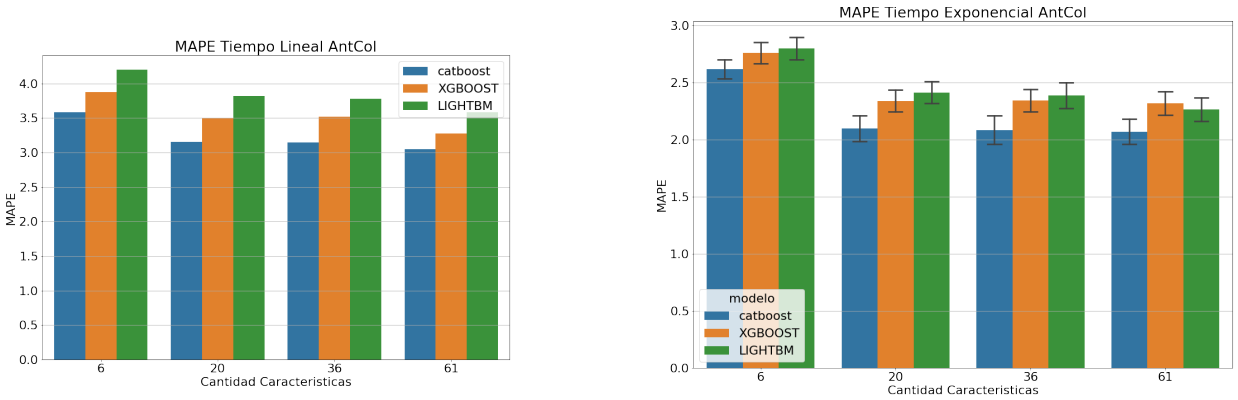
(b) GM Sin Head y HybridEA los solvers 10-CV variando cantidad de características

Fig. 4.20: Métrica GM obtenida por los modelos de clasificación sobre el conjunto de validación con 10-CV

Los resultados reportados en las Figura 4.20a presentan una amplia variación tal como sucedía en la tablas 4.9, 4.10. La razón principal es el desbalanceo de clases y al realizar la separación entre conjunto de entrenamiento y validación, se tiene que algunas clases quedan pobremente representadas en algunos de los conjuntos. Como evidencia de esto se puede ver la Fig. 4.20b, donde esta variación de la métrica disminuye, ya que hay una mejor representación de las clases aunque siguen existiendo algunas que tiene pocos casos como *Sin solución*, *Color6* y *pop2*, esto puede ser evidenciado en la tabla 4.7.

Por otra parte, para la regresión se analizan ambos tiempos, en este caso solo se analizó el conjunto de datos del solver *AntCol*, usando tres regresores como base: LightGBM, Catboost y XgBoost. De las Fig. 4.21a y Fig. 4.21b se concluye que en ambos casos Catboost obtiene el menor error para la métrica MAPE en la curva de tiempo *exponencial*, también se observa que a medida que se incorporan características, el error disminuye, aunque las diferencias del rendimiento obtenido se hacen mayormente notorias entre 6 y 20 características.

Para realizar una evaluación se definió un conjunto de test, que corresponden a 1314



(a) Lineal

(b) Exponenciales

Fig. 4.21: Métrica MAPE obtenida por los modelos de regresión sobre el conjunto de validación con 10-CV

instancias que no se encuentran en el conjunto de entrenamiento. Este conjunto esta compuesto por 104, 724, 585 instancias Trivial, Easy y Hard, respectivamente. Además tiene una distribución de clases que puede ser vista en la tabla 4.11.

Algoritmo	Proporción
Head	0.824779
HybridEA	0.082919
AntCol	0.036856
HillClimber	0.022022
BacktrackingDSatur	0.021400
Color6	0.010578
Sin Solución	0.000619
Pop2	0.000587
PartialCol	0.000240

Tabla 4.11: Distribución clases en conjunto test

4.5. Mejores modelos

De lo anterior se selecciona el mejor modelo obtenido utilizando la menor cantidad de características y se realiza una optimización de hiper-parámetros usando Hyperopt [101], una

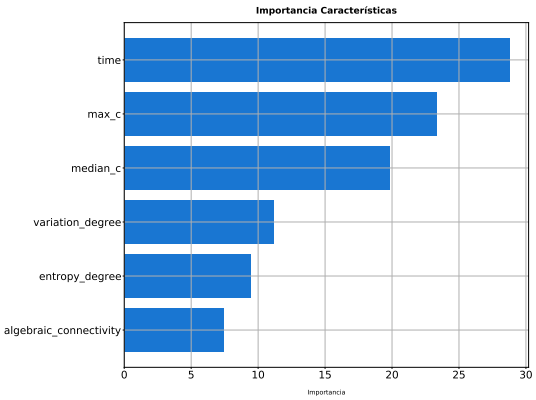
librería de optimización bayesiana que usa el algoritmo Tree Parzen Estimator [117]. Por cada iteración se hacen 3 validaciones cruzadas. Se aplicó un detector de sobreajuste, realizando aproximadamente 150 iteraciones del optimizador por modelo. Este ajuste de modelos fue realizado usando Google Colab. Esta plataforma permite el uso de GPU; de esta manera un modelo que puede tomar 1 hora en entrenar en CPU puede demorar 10 minutos en GPU, lo que permite realizar varias iteraciones de la búsqueda en una menor cantidad de tiempo. Una de las desventajas actuales de la plataforma radica en su cantidad de memoria RAM, lo cual afectó en el entrenamiento de los modelos de regresión.

4.5.1. Clasificación

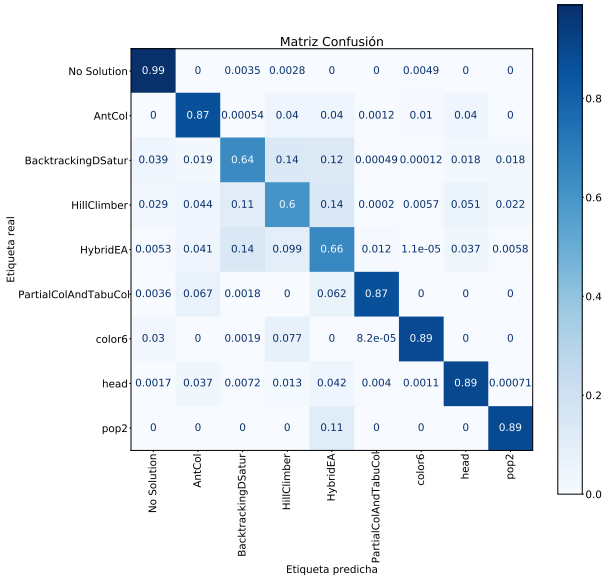
Para este caso se realizaron pruebas con los modelos Xgboost y Catboost, aunque finalmente se decidió por Catboost debido a que los ajustes toman un menor tiempo en comparación con XGboost.

El modelo obtiene un puntaje medio de $0{,}7072 \pm 0{,}12$ en la métrica media geométrica haciendo 5 validaciones cruzadas sobre el conjunto de entrenamiento, lo cual es una mejora con respecto a los resultados de la tabla 4.10 tanto en la media como en la desviación estándar. En este modelo las clases con menor presencia como *PartialCol*, *Pop2* y *Sin solución* son predichas de mejor manera. Esto se puede concluir luego de revisar la matriz de confusión (anexo A.3) se ve que las 3 clases anteriormente mencionadas obtiene peores resultados y en la Fig. 4.22b, se puede notar que se logra predecir mejor las clases. En ambos casos se puede intentar mejorar las predicciones en las clases *BacktrackingDsatur*, *HillClimber* y *HybridEA*.

De la Fig. 4.22a, se puede concluir que la característica más importante es el tiempo y que a diferencia de lo ocurrido en la Fig. 4.19, las demás características utilizadas adquieren una mayor importancia en los resultados. Para los modelos las características más relevantes fueron la variación y la entropía de los grados de los nodos, el número máximo obtenido por el coeficiente de clustering, la variación entre los pares de distancia y el segundo valor propio mínimo de la matriz de adyacencia.



(a) Importancia característica

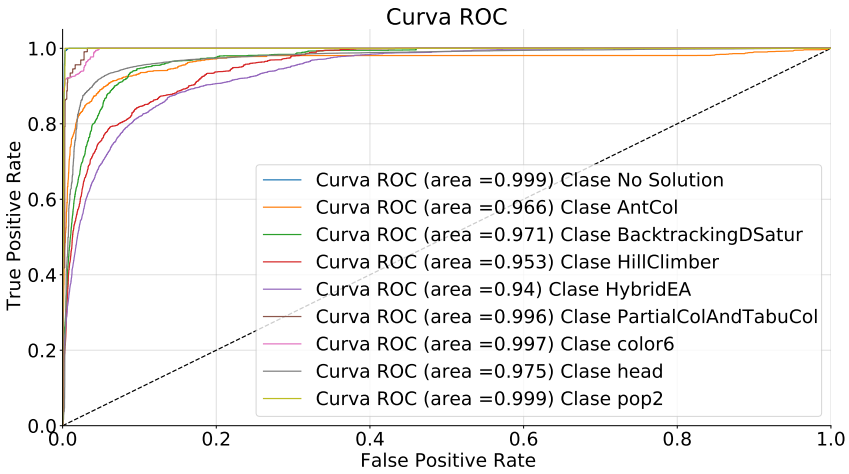


(b) Matriz de confusión

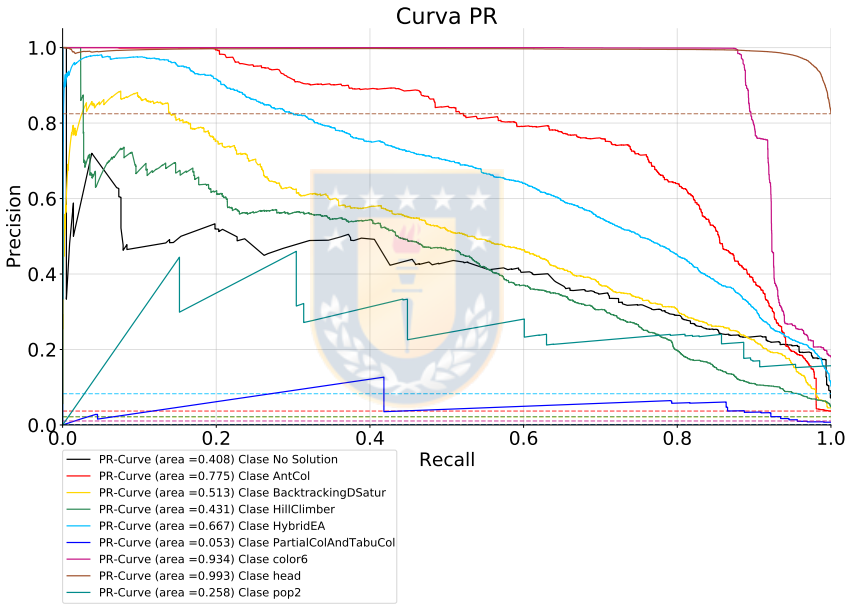
Fig. 4.22: Matriz de confusión e importancia de características sobre conjunto de test

Si se desea evaluar el modelo basado en qué tan efectivo es para separar las clases, es posible utiliza la curva ROC que resume la habilidad del modelo para reconocer las clases, la cual puede ser vista en Fig.4.23a, de aquí se puede observar que todas las clases se reconocen bien. Pero en algunas clases se tienen pocos datos lo cual puede afectar en que la métrica sea engañosa.

Como método alternativo es posible evaluar la curva Precision-Recall, que permite valorar la calidad del modelo cuando las clases se encuentran muy desbalanceadas. Considerando esta diferencia hay que contextualizar la métrica, calculando el porcentaje de presencia de la clase en el conjunto de test (Tabla 4.11) y compararla con el área bajo la curva obtenida desde la Fig. 4.23b. Con esto se puede tener una idea de qué tan bien se logran predecir las clases comparado con un modelo sin habilidad. De aquí se observa que *PartialCol* y *Sin solución*, son las clases que se predicen peor. La mejor combinación de parámetros sobre el conjunto de validación obtuvo las métricas de la tabla 4.12 sobre el conjunto de test.



(a) Curva ROC



(b) Curva precision-recall

Fig. 4.23: Curva ROC y Curva PR sobre conjunto de test

Metrica	Valor
accuracy	0.861
recall_weighted	0.861
f1_weighted	0.878
precision_weighted	0.908
cohen_kappa_score	0.63
matthews_corrcoef	0.645
gm	0.8
gm_macro	0.892
gm_micro	0.92
gm_weight	0.912

Tabla 4.12: Métricas mejor clasificador sobre conjunto de test

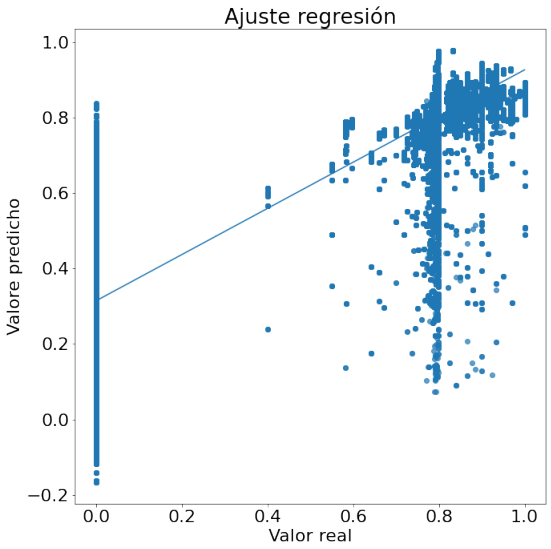
Alternativamente y para comprobar la validez de la metodología se entrenaron modelos siguiendo una curva de tiempo lineal haciendo variar los solvers a utilizar. En el anexo A.4, están los resultados de quitar el solver *Head* del conjunto de solver. De este modelo se puede decir que *HybridEA* es el solver predominante, porque lo que se sigue teniendo un desbalanceo de clases, tal como se ve en la tabla 4.7. En anexo A.5, se muestran los resultados de quitar del conjunto de solvers *Head* y *HybridEA*, donde se presenta un escenario de mayor competencia.

4.5.2. Regresión

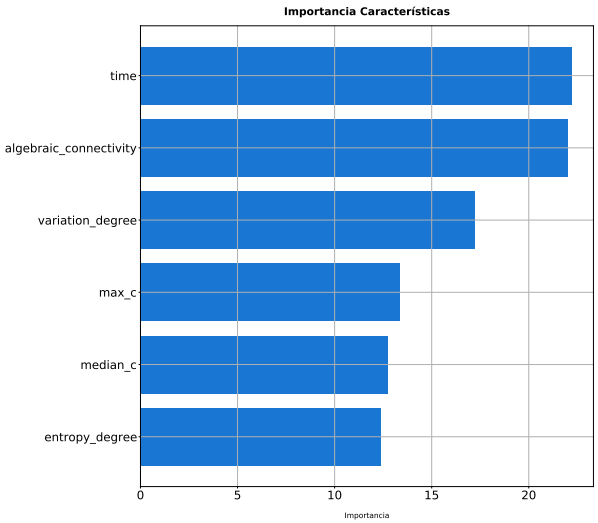
Para este caso se entreno Catboost que obtiene los mejores resultados en las figuras Fig. 4.24a y Fig. 4.24b; se intento realizar un ajuste de hiperparámetros usando el tiempo exponencial. Sin embargo, debido al alto uso de memoria RAM no fue posible, por lo que se decidió ajustar el tiempo lineal en su reemplazo. Para el entrenamiento de este modelo se usaron las mismas 6 características utilizadas en clasificación, ya que de esta manera cuando se realice una comparación entre ambos modelos es posible simplificar el análisis.

En la Fig. 4.24a se puede ver que el modelo tiene dificultad para predecir los valores cercanos a 0 y los valores cercanos a 0.8 principalmente; donde entre 0.8 y 1 se puede observar una agrupación de datos. De la Fig. 4.24b se observa que se obtienen distintos valores de

importancia con respecto a lo reportado en la clasificación, manteniéndose que la característica más importante es el tiempo.

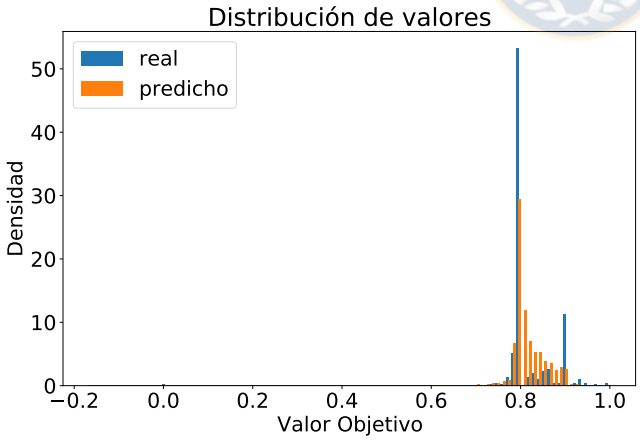


(a) Ajuste mejor modelo

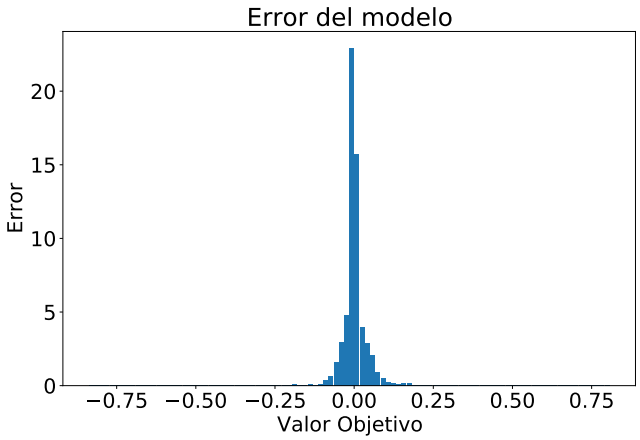


(b) Importancia características

Fig. 4.24: Modelos regresión tiempos



(a) Distribución predicciones



(b) Error modelo regresión

Fig. 4.25: Error modelo regresión

Si bien el análisis anterior es realizado sobre los resultados obtenidos por *AntCol*, se puede notar que el comportamiento de los modelos en base a las métricas de la tabla 4.13 son similares entre sí.

Lineal	AntCol	BacktrackingDsatur	HillClimber	HybridEA	PartialCol	Color6	Head	pop2
MAPE	2.603	2.603	2.596	2.623	2.672	2.612	2.679	2.606
MAE	0.021	0.021	0.021	0.022	0.022	0.021	0.022	0.021
RMSE	0.043	0.044	0.043	0.043	0.044	0.043	0.044	0.044
R2	0.587	0.567	0.573	0.576	0.566	0.57	0.564	0.564

Tabla 4.13: Métricas regresión obtenidas por cada solver

Por un lado se tiene la métrica MAPE, que indica el error en las unidades de medida del problema, por lo que en promedio los modelos presentan un error del 3 % de CMSE; el problema de esta métrica, es que no es posible determinar si el error es predecir de más o de menos. Sin embargo, en la Fig. 4.24a y la Fig. 4.25b se permite observar que se producen ambos casos. De la Fig. 4.25a se puede apreciar que el modelo genera una distribución similar, pero existen casos donde el modelo retorna valores cercanos a 0 y éstos no se notan presentes en los datos a predecir. Los modelos de regresión logran predecir con cierta dificultad el CMSE de cada solver. Sin embargo, cada modelo por separado, podría presentar utilidad cuando se quiere estimar el rendimiento del solver sin ejecutar la instancia.

En general de los enfoques anteriores en ambos casos es posible construir un modelo. Si bien el problema del desbalanceo representa un reto para los modelos de clasificación, es posible realizar un ajuste de parámetros para obtener un buen rendimiento en base a la métrica de media geométrica. Aunque se debería evaluar el hecho de eliminar aquellos solvers que tiene poca presencia de tiempos ganadores, ya que demuestran ser poco competitivos contra los demás. Si bien se logra obtener buenas métricas para la clasificación, se puede concluir de la validación cruzada que la métrica GM tienen una gran variabilidad, producida por el mismo desbalance, ya que en algunas iteración se cuenta con 5 ejemplos por clase. Como solución a lo anterior se debe mejorar la proporción de instancias donde solvers distintos a Head y HybridEA sean ganadores, aunque esto podría no ser posible, ya que aún en nuevas instancias puede que estos solvers sean los predominantes.

En el caso de la regresión, todos los modelos entrenados por conjunto de datos por solver obtienen métricas similares. Aunque el tiempo de predicción elevado comparado al de clasificación puede ser una clara desventaja cuando lo que se procura disminuir sea el tiempo.

5. Resultados

En este capítulo se evalúa cuál de los enfoques logra representar con mayor acierto la situación. Para esto se genera un escenario tipo competencia para evaluar la utilidad práctica de los modelos. Para esto se dividió en dos partes: Primero se compara lo obtenido por el modelo de clasificación con el regresor y luego se compara el mejor modelo obtenido en la primera etapa con la ejecución de los solvers. Para realizar las evaluaciones, se debe definir *el tiempo de predicción*. *Tiempo Predicción = Tiempo Consultado - (Tiempo Calculo Características + Tiempo Predicción Modelo)*. De esta manera, se le asigna un punto al modelo si es capaz de entregar el mejor solver en un tiempo determinado sin superar el tiempo de predicción. Así, se debe cumplir que el tiempo de predicción debe ser mayor a 0 y el modelo debe entregar una solución mejor o igual que la de los solvers por separado.

Para esto se debe considerar que la mayoría de los cambios de solver ganador suceden en los primeros instantes de ejecución de los solvers, pero debido al cálculo de las características sería injusto realizar evaluaciones en los primeros instantes, ya que en muchos casos éstas no se alcanzan a calcular. En detalle se tiene que del total de instancias, el 50 % de ellas calcula sus características a los 429 ms mientras que el 70 % de ellas a los 1805 ms (aproximadamente 2 segundos), en ambos casos si se considera la cantidad de tiempos analizados, ambos no sobrepasan el 1 % presentes en la curva de tiempo. Por ende, considerando la información anteriormente mencionada, el cálculo de características debe ser mejorado.

Si se considera una situación real, la utilidad de estos modelos recae en tiempos del orden de minutos, y no en segundos, que es donde se encuentra la mayor variabilidad. Por lo que, la utilidad del modelo va a depender principalmente de los tiempos en los cuales algún solver no sea considerado como dominador, ya que, en el caso de existir un solver dominador, el modelo debe predecir mejor aquellos tiempos donde no existe un solver dominador. Las pruebas han demostrado que el solver *Head* es el ganador en la mayoría de los tiempos. Para generar evaluaciones sobre los modelos, éstas son realizadas desde los 429 ms hasta los 44 minutos de ejecución, ya que este es el último tiempo en el que, en todas las instancias recolectadas, dejan de suceder cambios en el solver ganador.

5.1. Clasificador v/s Regresor

En un escenario de competencia, interesa considerar el tiempo que le toma al modelo realizar su predicción para elegir el de menor tiempo. En este sentido el modelo de clasificación tarda $1{,}68$ segundos en predecir todos los tiempos de las 1314 instancias del conjunto de test, y el modelo de regresión tarda $67{,}12$ segundos. En vista de los anterior se evaluaron métricas similares a la utilizadas en una competencia como *accuracy* y *recall* ambos modelos obtienen resultados similares, 0.861 para el clasificador y 0.79 para el regresor, en ambas métricas. Considerando el tiempo de predicción y los resultados sobre las métricas, el modelo de clasificación es superior al de regresión, y por ende, este ultimo es descartado.

Para realizar un mejor análisis de los resultados se presentan las siguientes métricas, que son más adecuadas para un entorno de competencia. Para esto solo se considera el modelo de clasificación, donde se analizan todos los tiempos y las instancias en el conjunto de test, sin considerar el *tiempo de predicción*, ya que el objetivo es entender el rendimiento mediante métricas tales como: la cantidad de lugares/posiciones al mejor solver sin considerar empates, otra que sí los considere y una métrica que considere el CMSE de la soluciones. De las ideas anteriores se generan 3 métricas, para las cuales es necesario definir un ranking de las soluciones.

Sea un tiempo $t \in T$, con T el conjunto de los tiempos generados, sea un solver $s \in A$, con A el conjunto de los solvers ordenados crecientemente de manera alfabética y una instancia $i \in P$ con P el conjunto de las instancias. Sea Sol_{it} un arreglo con n, la cantidad de solvers a usar que contiene las soluciones de los solvers para el instante t y una instancia i. Cada posición en el arreglo representa un solver, enumerados desde 0 a $n-1$ y ordenados de la misma manera que A. Se puede definir un orden creciente de las soluciones, para tener un ranking de posiciones que no permita empates, llamado ORD_{it}.

$$ORD_{it} = argsort(Sol_{it})$$

Además se debe definir un ranking que considere el empate de las soluciones, denominado $Rank_{it}$ con una instancia i y un tiempo t. Considerando el arreglo Sol_{it}, el cual se debe ordenar de manera creciente y considerar los valores únicos, para poder así generar un mapeo $(valor, posición)$ denominado pos_{it} de los valores en Sol_{it}. De esta manera para generar $Rank_{it}$ se debe mapear cada valor en Sol_{it} a la transformación pos_{it}.

- **Métrica 1**: Para esta métrica se necesita ordenar las posiciones de los solvers en función de sus soluciones, las cuales se ordenan de menor a mayor usando ORD_{it}. De esta manera el error de un modelo está dado por la distancia en posiciones con respecto al índice del solver predicho por el modelo ($Solver_{it}$) con la mejor solución para esa instancia y el tiempo consultado. Para esto el modelo debe entregar el índice del solver que obtiene la mejor solución, y de esta manera contar las posiciones de diferencia con respecto a ORD_{it}. Luego se divide el error en la cantidad de solvers presentes en el conjunto de solvers y se reporta la media de los errores. Así la métrica queda definida entre 0 y 1, siendo 0 el mejor valor y 1 el peor valor.

$$métrica\ 1_{it} = \frac{argwhere(ORD_{it}, Solver_{it})}{n}$$

- **Métrica 2**: Para esta métrica se necesita utilizar el ranking con empate $Rank_{it}$. Así, el error de un modelo esta dado por la distancia en posiciones con respecto a la mejor solución para esa instancia y el tiempo consultado. Para esto cada modelo debe entregar el índice del solver con la mejor solución ($posición_{it}$), así se puede consultar su ranking en $Rank_{it}$. Esta métrica es interesante, debido a que relaja la restricción de obtener solo un solver ganador. Como puede ser visto en las Fig. 4.13, Fig.4.15, Fig.4.14, y Fig.4.16, en muchos tiempos se producen empates. La métrica queda definida entre 0 y 1, siendo 0 el mejor valor y 1 el peor valor. Para reportar esta métrica se calcula como la formula a continuación y se reporta la media.

$$métrica\ 2_{it} = \frac{posición_{it}}{Max\ posición_{it}}$$

- **Métrica 3**: En esta métrica, se considera el CMSE de la solución entregada por el modelo ($solución_{it}$). Esta métrica tiene valores entre 0 (mejor valor) y 1 (peor valor). Se calcula como:

$$métrica\ 3_{it} = 1 - CMSE(solución_{it})$$

Para poder reportar resultados globales de las métricas anteriores, se calcula el promedio de todos los tiempos y todas las instancias consultadas.

A continuación se muestran los resultados obtenidos por el modelo de clasificación contra predecir siempre el solver predominante en las distintas métricas sobre el conjunto de test. Se debe destacar que en estas métricas no se considera el tiempo de cálculo de las características:

Con respecto a la métrica 1, el modelo de clasificación obtiene un error de 0.101. En la Fig. 5.26 se muestra el promedio del error obtenido en las distintas instancias y a distintos tiempos. De aquí se observa que el mayor error se produce en las instancias *easy*, en ellas el modelo es capaz de ganar al solver predominante antes de los 285 ms, ya que luego de este instante el solver predominante es el ganador. El menor error se produce en las instancias *Hard*, donde independiente del tiempo el clasificador es el ganador. Para las instancias *triviales*, los modelos empatan en los instantes antes de los 45 ms, ya que luego de ese tiempo el solver predominante es el ganador. En general, la mayor cantidad de errores se producen en los tiempos iniciales. Como la métrica considera distancia a la solución correcta sin empates se tiene que el 0.101 representa la distancia promedio a la solución correcta.

En la métrica 2 se obtiene un 0.0383 de error. En la Fig. 5.27, al igual que en el caso anterior, se muestra el error promedio en los tiempos. Se obtienen resultados similares a la métrica 1, ya que, el error es más grande en las instancias *easy* y se concentra en los primeros instantes de tiempo. En este caso el clasificador obtiene mejores resultados que solver predominante para las instancias *Hard* y *easy*, pero para las instancias *triviales* existen tramos donde el clasificador es mejor, antes de los 180 ms, un tramo de empate desde los 255 a 519 ms y luego el solver predominante es mejor. En este caso a diferencia del anterior, se permiten empates de soluciones y por esa razón los errores son más bajos.

Finalmente, la métrica 3 tiene un 0.0443 de error. En la Fig. 5.28 se muestra el error promedio a distintos tiempos. Esta métrica muestra lo que le falta a la solución entregada por el modelo para alcanzar la mejor solución encontrada en base al CMSE. En las instancias *Hard* e *Easy* el modelo obtiene un menor error comparado con el solver predominante, pero en las instancias *triviales* el modelo es capaz de ganar hasta los 180 ms, ya que luego de ese instante el solver predominante es el ganador.

Los errores generales de las métricas separados por clases pueden ser visto en la tabla 5.14. Donde, en todos los casos, los errores son mayores en las instancias *easy*.

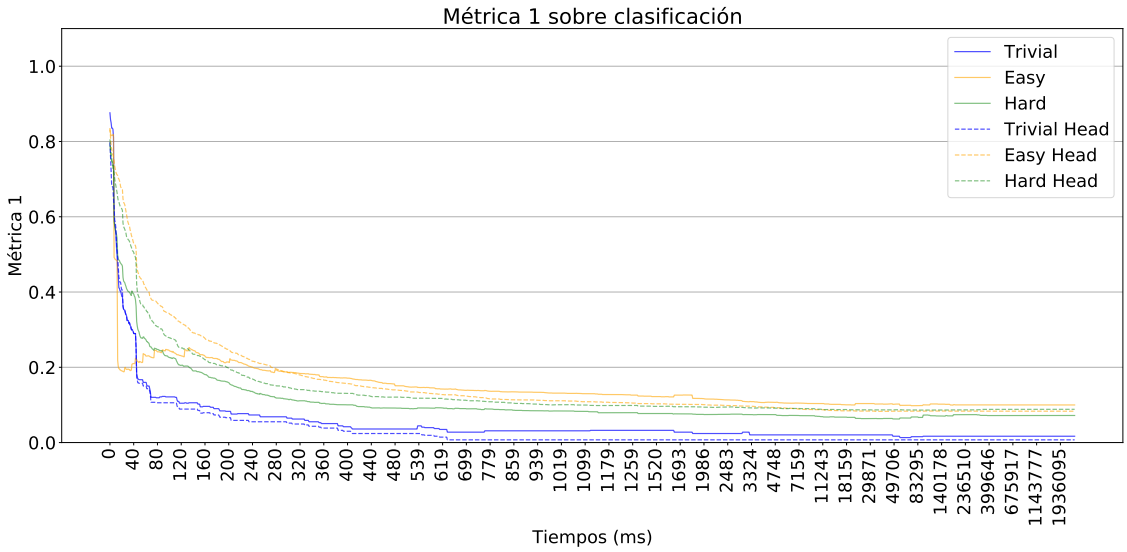


Fig. 5.26: Métrica 1 sobre clasificador

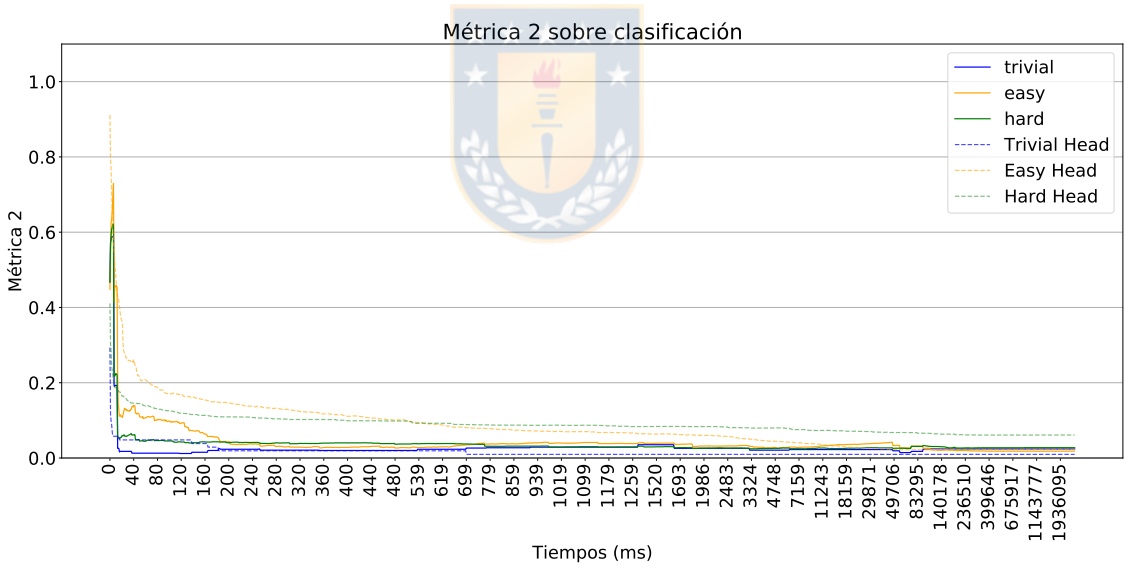


Fig. 5.27: Métrica 2 sobre clasificador

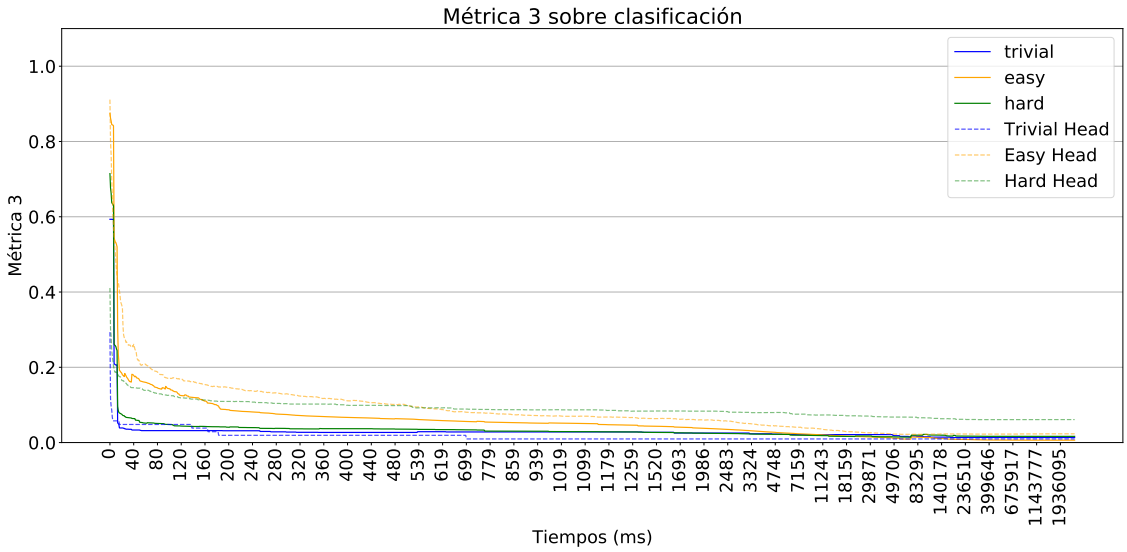


Fig. 5.28: Métrica 3 sobre clasificador

Métrica	Trivial	Easy	Hard	Global
Métrica 1	0.05	0.145	0.107	0.100
Métrica 2	0.026	0.0420	0.0360	0.0383
Métrica 3	0.0279	0.0566	0.0328	0.0443

Tabla 5.14: Resumen métricas obtenidas por el modelo de clasificación

5.2. Modelos v/s Solvers

Para realizar una evaluación del modelo contra la ejecución de los solvers por sí solo, se debe definir una métrica que ayude a entender la utilidad del modelo donde se considere el tiempo. Aquí el factor más importante a considerar es el cálculo de las características del grafo. De esta manera se puede calcular la proporción de aciertos por la cantidad de veces que gana o empata el clasificador dividido por la cantidad de instancias con los tiempo consultados. Así se compara el número de colores entregado por el modelo contra el menor número de colores obtenidos por los solvers, es decir, se consideran empates.

Para comparar ambos modelos, se utilizaron las instancias del conjunto de test. Los resultados obtenidos pueden ser divididos en dos: cuando no se considera el *Tiempo predicción*

y cuando sí se considera. Se consideró que el tiempo promedio en predecir es de 0.4 ms y el tiempo de cálculo de características varía de acuerdo a la instancia consultada.

Para considerar un tiempo inicial de consulta se debe tener en cuenta que el 50 % de las instancias calculan las características a los 429 ms, por lo que solo se consultaran tiempos superiores a estos para realizar esta evaluación. Las pruebas son realizadas con tiempos (timeout) 540, 2400, 3000, 6000, 7200, 9000, 10800, 12000, 24000, 60000, 900000, 1500000, 1800000 y 2640000 ms, de esta manera se puede considerar qué sucede en los primeros instantes después del cálculo de las características, en tiempos intermedios y tiempos finales.

En la tabla 5.15 se muestran los resultados de considerar todos los solvers, sin considerar *Head* y si se considera o no el *Tiempo de predicción*. Si no se, toma en cuenta el *Tiempo predicción*, sin *Head* se tiene un error de 0.15 %, y por el contrario, si se considera, resulta un error de 0.094 %. El error aumenta notoriamente en ambos casos cuando se tiene presente el *Tiempo de predicción*, ya que si consideramos *Head* tiene un error de 0.229 y si no uno de 0.29.

	No considera tiempo	**Considera tiempo**
Con Head	0.094	0.229
Sin Head	0.15	0.29

Tabla 5.15: Proporción de error del modelo comparado con ejecución de los solvers

Para analizar esto en mayor profundidad se evaluaron 3 posibles escenarios: utilizar todos los solvers, quitar *Head*, quitar *Head* y *HybridEA*. De estos 3 escenarios, existen 2 casos. Primero uno donde el solver predominante es demasiado dominador (*Head* y *HybridEA*), aquí existe un tiempo donde el solver predominante es capaz de ganar en la mayoría los casos. Como sucede en el escenario de todos los solver, donde en la tabla 5.16 se puede notar que existen momentos, principalmente los iniciales, hasta los 7200 ms donde el modelo es capaz de empatar o ganar. Contrariamente en el caso sin *Head* el modelo no es capaz de ganar en ningún instante de tiempo como se puede notar en la tabla 5.17. En el segundo caso, hay un escenario de mayor competencia, cuando se quita *Head* y *HybridEA* del conjunto. En este caso, el modelo es capaz de ganar al solver predominante independiente del *timeout*, como puede ser visto en la tabla 5.18.

Timeout (ms)	No considera tiempo	Considera tiempo	Diferencia	Head sin tiempo
540	0.886	0.452	0.435	0.824
2400	0.900	0.619	0.281	0.874
3000	0.893	0.643	0.250	0.884
6000	0.895	0.713	0.182	0.895
7200	0.900	0.728	0.171	0.900
9000	0.895	0.742	0.153	0.905
10800	0.887	0.746	0.141	0.910
12000	0.888	0.750	0.138	0.916
24000	0.878	0.793	0.085	0.930
60000	0.902	0.846	0.056	0.940
900000	0.938	0.937	0.001	0.943
1500000	0.938	0.938	0.000	0.943
1800000	0.938	0.938	0.000	0.943
2640000	0.938	0.938	0.000	0.943

Tabla 5.16: Proporción de acierto solución todos los solvers.

Timeout (ms)	No considera tiempo	Considera tiempo	Diferencia	HybridEA sin tiempo
540	0.852	0.403	0.449	0.866
2400	0.855	0.568	0.287	0.870
3000	0.845	0.588	0.257	0.868
6000	0.846	0.668	0.178	0.868
7200	0.842	0.671	0.171	0.870
9000	0.842	0.688	0.154	0.875
10800	0.846	0.691	0.155	0.870
12000	0.834	0.684	0.150	0.873
24000	0.834	0.740	0.095	0.873
60000	0.839	0.815	0.024	0.915
900000	0.850	0.849	0.001	0.937
1500000	0.850	0.850	0.000	0.938
1800000	0.849	0.849	0.000	0.937
2640000	0.849	0.849	0.000	0.938

Tabla 5.17: Proporción de acierto solución sin Head.

Timeout (ms)	No considera tiempo	Considera tiempo	Diferencia	Partialcol sin tiempo
540	0.530	0.235	0.295	0.457
2400	0.605	0.372	0.233	0.525
3000	0.625	0.399	0.226	0.534
6000	0.654	0.504	0.150	0.531
7200	0.652	0.510	0.142	0.537
9000	0.672	0.515	0.158	0.547
10800	0.675	0.524	0.151	0.553
12000	0.676	0.524	0.152	0.553
24000	0.692	0.594	0.098	0.566
60000	0.689	0.667	0.023	0.585
900000	0.662	0.662	0.001	0.611
1500000	0.663	0.663	0.000	0.612
1800000	0.663	0.662	0.001	0.611
2640000	0.662	0.662	0.000	0.614

Tabla 5.18: Proporción de acierto solución sin Head y HybridEA.

De lo anterior, es posible notar que, en general, mientras más avanza el tiempo, el solver dominante se vuelve aun más dominante y que ha medida que continúa el tiempo, el efecto del cálculo de características va disminuyendo, donde en los instantes finales de la competencia ya deja de tener un efecto considerable desde los 900000 ms. Mientras exista un solver dominante en distancia largas el problema se vuelve mas difícil y el *tiempo de predicción* pierde importancia.

Tal como, puede ser, visto en las tablas anteriores, el cálculo de las características toma tiempo, y esto hace que al realizar una competencia entre usar el modelo y el solver predominante, el modelo corre en desventaja cuando se considera el tiempo. Esto sucede ya que si se elige el solver predominante para todos los tiempos, se obtienen mejores resultados en la competencia, pero desde el punto de vista de machine learning, no se cumple el objetivo de entrenar un modelo que sea capaz de aprender a reconocer en que instante de tiempo un solver es más adecuado que otro. De aquí es posible notar que el entorno donde se desea ejecutar el modelo es desafiante. Además de las tablas anteriores es posible concluir que el modelo es capaz de aprender.

Finalmente, es posible concluir que el modelo de clasificación logra representar la situación y ser competitivo contra la ejecución de los solvers por separado, pero podría ser mejorable

si fuera posible reducir el tiempo tomado en el cálculo de las características o si se eliminan la necesidad de ellas. Las métricas reportadas por esta evaluación varían según el conjunto de solvers a considerar; en ambos casos cuando se considera *Head* y cuando no, el modelo presenta un error de 20-30 % aproximado con respecto a la ejecución por separado. El error es mayor en las instancias *easy*, esto puede deberse a que en esta clase es donde los solvers más desfavorecidos salen más veces ganadores. Además de la comparación contra predecir siempre el solver predominante, es posible notar que no existe una gran diferencia entre el modelo y usar siempre el solver predominante, ya que como se ha dicho anteriormente el solver predominante obtiene las mejores soluciones para gran parte de las instancias y los tiempos considerados.

Por su lado, el modelo de regresión logra predecir el CMSE de cada modelo, pero al realizar la modificación para ser llevado a competencia, se observa que el modelo resultante no entrega buenos resultados comparado a lo obtenido por el modelo de clasificación. Además, tiene la desventaja de que, por su arquitectura, predecir toma mucho tiempo comparado con el de clasificación y como se ha recalcado anteriormente los ahorros de tiempo son necesarios. Es importante destacar que en un entorno de competencia de solver donde existe un solver dominador a lo largo de todos los tiempos y dataset, generar modelos de meta-aprendizaje no resulta conveniente, pero en escenarios de mayor competencia estos modelos pueden aprender y ser útiles.

6. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se propuso un nuevo enfoque de selección automática de algoritmos, centrado en el comportamiento anytime de los distintos enfoques de solución. Este nuevo enfoque fue estudiado a través de su aplicación al problema de Coloreo de Grafos.

Para la realización de este trabajo se recolectaron instancias y solvers propuestos en la literatura y se modificaron estos últimos para obtener su comportamiento anytime. Con respecto a las instancias, se utilizaron benchmarks públicos, tanto pertenecientes a aplicaciones de problemas reales, como creados a través de generadores de grafos. Entre estos últimos destaca el uso de HiSampler, que genera instancias difíciles para solvers específicos, utilizando técnicas de aprendizaje. Debido a la necesidad de obtener el código fuente de los solvers para extraer su comportamiento anytime, este trabajo se limitó únicamente a aquellos solvers cuyo código fuente es de acceso público. Se intentó conseguir el código fuente de otros solvers de la literatura, mediante comunicación directa con sus autores, sin recibir respuesta.

La generación de datos para el desarrollo de esta tesis se realizó mediante la ejecución de todos los solvers del portafolio en todas las instancias. Los datos generados fueron posteriormente normalizados y analizados en función a dos curvas de tiempo: *lineal* y *exponencial*. A partir de este análisis, se determinó que la curva lineal captura más información y se ajusta de mejor manera a la tarea. A partir de estos datos normalizados, fueron generados 2 enfoques de aprendizaje distintos: uno de clasificación, que busca etiquetar, en base a la entrada, el solver más prometedor y otro de regresión que predice, para cada solver, el rendimiento esperado (medido a través de una métrica de nuestra autoría: CMSE).

Un subproducto muy importante de este trabajo puede ser observado en las Fig. 4.14 a la Fig. 4.16, que presentan a la comunidad de investigación una fotografía actualizada del estado del arte en el problema de coloreo de grafos. Estas imágenes evidencian de manera gráfica el comportamiento anytime promedio de los solvers. Esta evaluación experimental y su forma de presentación es aplicable a otros dominios. En esta línea, se destacan los resultados obtenidos por el solver *Head*, que adquiere las mejores soluciones para gran parte de las instancias utilizadas, seguido, con bastante distancia, por el solver *HybridEA*.

Para el desarrollo de este trabajo, se puso bastante esfuerzo en el estudio de las características a proveer a los modelos de Machine Learning. En este sentido, se pudo identificar un

conjunto suficiente (mínimo) de características, capaces de representar a los distintos tipos de grafos. A pesar de haber reducido significativamente el número de estas características, se pudo observar que el tiempo de cómputo de las mismas no es menor y podría tener una influencia importante en la utilidad práctica de este tipo de enfoques. Como era esperable al inicio de esta investigación, se comprobó que la característica más representativa es el tiempo límite de ejecución, lo que sustenta la importancia de incorporar el comportamiento anytime de los solvers al proceso de selección automática de algoritmos. En esa misma línea, la metodología aquí planteada puede extenderse para considerar más restricciones sobre los recursos computacionales disponibles, como: cantidad de procesadores, cantidad máxima de Random Access Memory (RAM) a utilizar, permitir el uso de paralelismo en la ejecución de los solvers, etc.

Con respecto a la identificación de métricas de evaluación significativas para la tarea, este trabajo presenta un estudio comparativo de una diversidad importante de indicadores. En este sentido, en relación a la métricas relacionadas a tareas de clasificación, una conclusión importante tiene que ver con la pertinencia de tomar la métrica MCC como guía del modelo. Pese a que, a priori, la optimización de esta métrica puede generar mejores resultados absolutos (i.e. predecir mejor qué solver obtendrá una mejor solución en el tiempo consultado), los modelos tienden a concentrar su aprendizaje en la clase predominante y generar errores mucho mayores en las demás clases. Para solucionar este sesgo, se observó que la métrica GM permite obtener un modelo balanceado que logra distinguir entre todas las clases. Esto se ve reflejado en la matriz de confusión y en la curva de Precision-recall. Desde un punto de vista de Ciencia de Datos, este último constituye un modelo más correcto y útil ante una mayor generalidad de escenarios. Por otro lado, el modelo de regresión obtiene métricas más o menos similares para todos los solvers y evidencia dificultades para predecir valores cercanos al 0 y al 0.8.

De cara al objetivo central de este trabajo, si se evalúan ambos tipos de modelos (regresión y clasificación), se concluye que el modelo de clasificación es el más adecuado y los resultados finales, en un entorno tipo competencia, con este modelo, nos permiten concluir lo siguiente. Si se descarta el *tiempo de predicción* el modelo, según el escenario que se considere, es capaz de superar el rendimiento del mejor solver promedio del conjunto. En el escenario de considerar todos los solvers del portafolio, el modelo es capaz de empatar/ganar antes de los 7200 ms. En el caso de prescindir del solver *Head*, el modelo no es capaz de ganar a *HybridEA*, en cambio en el escenario, donde *Head* y *HybridEA* no son considerados, el modelo es capaz de ganar independiente del tiempo límite seleccionado. Estos resultados son distintos en el caso (más cercano a la aplicación) de considerar el tiempo necesario para la predicción. En este

caso, al menos para los primeros segundos, el modelo no muestra ser competitivo. A medida que se consideran tiempos límite superiores, el tiempo de cálculo de características y predicción se hace despreciable y el modelo se vuelve más útil (en los escenarios en los que existe mayor competencia entre los solvers).

El conjunto de solvers utilizados para la realización de este trabajo presenta un desafío mayúsculo desde el punto de vista de machine learning, ya que existen métodos de solución fuertemente dominantes y son pocos los escenarios en los que un clasificador / regresor, podrá seleccionar un algoritmo distinto. Por el mismo motivo, la aplicación práctica de enfoques como el que se propone aquí o, de forma general, cualquier enfoque de selección automática de algoritmos, es limitada. Sin embargo, el desarrollo mostrado en este documento muestra un potencial muy grande de aplicación y efectividad, en escenarios en los que no existe una dominancia tan clara. La metodología aquí presentada es agnóstica a los solvers específicos del portafolio, así como a los benchmarks concretos sobre los que se realiza el aprendizaje. Por tanto, el marco de trabajo aquí presentado podría ser fácilmente utilizado en un futuro próximo ante la aparición de nuevos solvers, la existencia de un mayor número de instancias con base en problemas reales y, de forma general, mayor competencia y diversidad en el estado del arte.

Adicionalmente a la complejidad de lidiar con escenarios fuertemente desbalanceados, también constituye un desafío importante lidiar con el efecto de considerar el tiempo de computo de las características y el tiempo de ejecución del modelo de predicción. Para una utilización práctica de este enfoque, ese tiempo tiene que ser considerado en la investigación, lo que demanda mayor trabajo en formas de caracterizar los grafos y optimizar al máximo el funcionamiento de los modelos. Posibles alternativas para lo primero es el uso de redes convolucionales de grafos [118] o utilizar características dinámicas de los primeros instantes de ejecución de los solvers.

Todos los datos generados en esta investigación han sido dispuestos de forma pública a manera de plantear un nuevo desafío para la comunidad de machine learning, así como para estimular a la comunidad científica relacionada con el Coloreo de Grafos para generar nuevos enfoques o mejorar los propuestos en este trabajo. En particular, hasta donde hemos podido conocer, este es el primer trabajo que propone considerar el comportamiento anytime de los métodos de solución en la investigación relacionada con la selección automática de algoritmos de coloreo de grafos.

Glosario

AMPaX Adaptive Multi-Parent crossove

CDCL Conflict-Driven Clause Learning

CEGAR Counter-Example Guided Abstraction Refinement

CMSE Cercanía a la Mejor Solución Encontrada

CNF Conjunctive Normal Form

CSP Constraint Satisfaction Problem

CV Cross validation

DSATUR Degree Saturation

FPR False Positive Rate

GA Algoritmo Genético

GCP Graph Coloring Problem

GM Geometric Mean

GOSS Gradient-based One-Side Sampling

GPX Greedy Partition Crossover

HEA Hybrid Evolutionary Algorithm

KNN K-Nearest Neighbors

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

MCC Coeficiente de correlación de Matthews

MIS Maximal Independient Set

ms milisegundos

MVS Minimal Variance Sampling

NAN Not A Number

NFL No Free Launch

PCA Principal Component Analysis

RAM Random Access Memory

RFE Recursive Feature Elimination

RLF Recursive Largest First

RMSE Root Mean Square Error

ROC Receiver Operating Characteristic

SAT Boolean Satisfiability Problem

SMBO Sequential Model-Based Optimization

SOFM Self Organization Feature Maps

SVM Support Vector Machine

TPE Tree Parzen Estimator

TPR True Positive Rate

Referencias

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Trans. Evol. Comp*, vol. 1, no. 1, p. 67–82, Apr. 1997. [Online]. Available: https://doi.org/10.1109/4235.585893

[2] J. C. Culberson, "On the futility of blind search: An algorithmic view of "no free lunch"," *Evolutionary Computation*, vol. 6, no. 2, pp. 109–127, 1998.

[3] J. R. Rice, "The algorithm selection problem." *Advances in Computers*, vol. 15, pp. 65–118, 1976. [Online]. Available: http://dblp.uni-trier.de/db/journals/ac/ac15.html#Rice76

[4] E. A. Hansen and R. Zhou, "Anytime heuristic search," *CoRR*, vol. abs/1110.2737, 2011. [Online]. Available: http://arxiv.org/abs/1110.2737

[5] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[6] M. Gamache, A. Hertz, and J. Ouellet, "A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding," *Computers & OR*, vol. 34, pp. 2384–2395, 08 2007.

[7] N. Zufferey, P. Amstutz, and P. Giaccari, "Graph colouring approaches for a satellite range scheduling problem," *Journal of Scheduling*, vol. 11, 08 2008.

[8] V. Lotfi and S. Sarin, "A graph coloring algorithm for large scale scheduling problems," *Comput. Oper. Res.*, vol. 13, no. 1, p. 27–32, Jan. 1986. [Online]. Available: https://doi.org/10.1016/0305-0548(86)90061-4

[9] D. de Werra, "An introduction to timetabling," *European journal of operational research*, vol. 19, no. 2, pp. 151–162, 1985.

[10] E. Burke, B. MacCloumn, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper heuristic for timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177–192, 06 2007.

[11] D. de Werra, C. Eisenbeis, S. Lelait, and B. Marmol, "On a graph-theoretical model for cyclic register allocation," *Discrete Applied Mathematics*, vol. 93, no. 2-3, pp. 191–203, 1999.

[12] T.-K. Woo, S. Y. Su, and R. Newman-Wolfe, "Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm," *IEEE Transactions on Communications*, vol. 39, no. 12, pp. 1794–1801, 1991.

[13] R. R. Lewis, *A Guide to Graph Colouring: Algorithms and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[14] A. Bondy and U. Murty, *Graph Theory*, ser. Graduate Texts in Mathematics. Springer London, 2011. [Online]. Available: https://books.google.cl/books?id=HuDFMwZOwcsC

[15] V. Paschos, "Polynomial approximation and graph-coloring," *Computing (Vienna/New York)*, vol. 70, 02 2003.

[16] P. Pardalos and J. Xue, "The maximum clique problem," *Journal of Global Optimization*, vol. 4, pp. 301–328, 04 1994.

[17] P. Formanowicz and K. Tanaś, "A survey of graph coloring - its types, methods and applications," *Foundations of Computing and Decision Sciences*, vol. 37, 09 2012.

[18] R. Venkatesan and L. A. Levin, "Random instances of a graph coloring problem are hard," in *STOC '88*, 1988.

[19] A. Björklund and T. Husfeldt, "Exact algorithms for exact satisfiability and number of perfect matchings," *Algorithmica*, vol. 52, pp. 226–249, 07 2006.

[20] A. Marie de Lima and R. Carmo, "Exact algorithms for the graph coloring problem," *Revista de Informática Teórica e Aplicada*, vol. 25, p. 57, 11 2018.

[21] S. W. Golomb and L. D. Baumert, "Backtrack programming," *J. ACM*, vol. 12, no. 4, p. 516–524, Oct. 1965. [Online]. Available: https://doi.org/10.1145/321296.321300

[22] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 7th ed. New York, NY, USA: McGraw-Hill, 2001.

[23] J. a. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, p. 506–521, May 1999. [Online]. Available: https://doi.org/10.1109/12.769433

[24] J. Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," *Frontiers in Artificial Intelligence and Applications*, vol. 185, 01 2009.

[25] T. Mostafaie, F. Khiyabani, and N. Navimipour, "A systematic study on meta-heuristic approaches for solving the graph coloring problem," *Computers & Operations Research*, p. 104850, 11 2019.

[26] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533 – 549, 1986, applications of Integer Programming. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0305054886900481

[27] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science (New York, N.Y.)*, vol. 220, pp. 671–80, 06 1983.

[28] P. Galinier and A. Hertz, "A survey of local search methods for graph coloring," *Computers & Operations Research*, vol. 33, pp. 2547–2562, 09 2006.

[29] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," 01 1991.

[30] P. Preux and E.-G. Talbi, "Towards hybrid evolutionary algorithms," *International Transactions in Operational Research*, vol. 6, no. 6, pp. 557 – 570, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0969601699000192

[31] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms," *Caltech Concurrent Computation Program*, 10 2000.

[32] F. Hayes-Roth, "Review of .ªdaptation in natural and artificial systems by john h. holland", the u. of michigan press, 1975," *SIGART Bull.*, no. 53, p. 15, Aug. 1975. [Online]. Available: https://doi.org/10.1145/1216504.1216510

[33] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. [Online]. Available: https://www.amazon.com/Machine-Learning-Pr

obabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2

[34] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, "Classification and regression trees," 1983.

[35] T. K. Ho, *Random Decision Forests*, 1995.

[36] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics and Data Analysis*, vol. 38, pp. 367–378, 1999.

[37] L. W. Pierre Geurts, Damien Ernst, *Extremely randomized trees*, 2006.

[38] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, p. 273–297, Sep. 1995. [Online]. Available: https://doi.org/10.1023/A:1022627411411

[39] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1, pp. 185 – 234, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370289900490

[40] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, pp. 427–437, 07 2009.

[41] G. Jurman, S. Riccadonna, and C. Furlanello, "A comparison of mcc and cen error measures in multi-class prediction," *PloS one*, vol. 7, p. e41882, 08 2012.

[42] R. Espíndola and N. Ebecken, "On extending f-measure and g-mean metrics to multi-class problems," *Sixth international conference on data mining, text mining and their business applications*, vol. 35, pp. 25–34, 01 2005.

[43] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2003.

[44] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology," 09 2018.

[45] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *CoRR*, vol. abs/1811.12808, 2018. [Online]. Available: http://arxiv.org/abs/1811.12808

[46] P. Brazdil and C. Giraud-Carrier, "Metalearning and algorithm selection: Progress, state of the art and introduction to the 2018 special issue," *Mach. Learn.*, vol. 107, no. 1, p. 1–14, Jan. 2018. [Online]. Available: https://doi.org/10.1007/s10994-017-5692-y

[47] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *CoRR*, vol. abs/1811.11597, 2018. [Online]. Available: http://arxiv.org/abs/1811.11597

[48] C. P. Gomes and B. Selman, "Algorithm portfolios," *Artificial Intelligence*, vol. 126, no. 1, pp. 43 – 62, 2001, tradeoffs under Bounded Resources. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370200000813

[49] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: Portfolio-based algorithm selection for SAT," *CoRR*, vol. abs/1111.2249, 2011. [Online]. Available: http://arxiv.org/abs/1111.2249

[50] K. Smith-Miles and L. Lopes, "Measuring instance difficulty for combinatorial optimization problems." *Comput. Oper. Res.*, vol. 39, no. 5, pp. 875–889, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/cor/cor39.html#Smith-MilesL12

[51] E. Horvitz and J. Breese, "Ideal partition of resources for metareasoning," 1999.

[52] D. Brélaz, "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, no. 4, p. 251–256, Apr. 1979. [Online]. Available: https://doi.org/10.1145/359094.359101

[53] V. Lotfi and S. Sarin, "A graph coloring algorithm for large scale scheduling problems," *Computers & Operations Research*, vol. 13, no. 1, pp. 27 – 32, 1986. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0305054886900614

[54] A. Hertz and D. Werra, "Werra, d.: Using tabu search techniques for graph coloring. computing 39, 345-351," *Computing*, vol. 39, 12 1987.

[55] K. A. Dowsland and J. M. Thompson, "An improved ant colony optimisation heuristic for graph colouring," *Discrete Appl. Math.*, vol. 156, no. 3, p. 313–324, Feb. 2008. [Online]. Available: https://doi.org/10.1016/j.dam.2007.03.025

[56] S. M. Korman, "The graph-colouring problem," *Combinatorial optimization*, pp. 211–235, 1979.

[57] Z. Zhou, C.-M. Li, C. Huang, and R. Xu, "An exact algorithm with learning for the graph coloring problem," *Comput. Oper. Res.*, vol. 51, p. 282–301, Nov. 2014. [Online]. Available: https://doi.org/10.1016/j.cor.2014.05.017

[58] E. Hebrard and G. Katsirelos, "Clause learning and new bounds for graph coloring," 08 2019, pp. 6166–6170.

[59] G. Glorian, J.-M. Lagniez, V. Montmirail, and N. Szczepanski, "An incremental sat-based approach to the graph colouring problem," 09 2019, pp. 213–231.

[60] R. Lewis, "A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing," *Computers & Operations Research*, vol. 36, pp. 2295–2310, 07 2009.

[61] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, pp. 379–397, 01 1999.

[62] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *European Journal of Operational Research*, vol. 203, pp. 241–250, 05 2010.

[63] L. Moalic and A. Gondran, "Variations on memetic algorithms for graph coloring problems," *Journal of Heuristics*, vol. 24, no. 1, p. 1–24, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s10732-017-9354-9

[64] I. Blöchliger and N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme," *Computers & Operations Research*, vol. 35, pp. 960–975, 01 2008.

[65] A. Jabrayilov and P. Mutzel, "New integer linear programming models for the vertex coloring problem," *CoRR*, 06 2017.

[66] NetworkX developer team, "Networkx," 2014. [Online]. Available: https://networkx.github.io/

[67] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, "A simple model to generate hard satisfiable instances," *IJCAI International Joint Conference on Artificial Intelligence*, 10 2005.

[68] R. Sato, M. Yamada, and H. Kashima, "Learning to find hard instances of graph problems," *CoRR*, vol. abs/1902.09700, 2019. [Online]. Available: http://arxiv.org/abs/1902.09700

[69] T. B. ThanhVu H. Nguyen. Graph coloring benchmark instances. [Online]. Available: https://mat.gsia.cmu.edu/COLOR08/

[70] "Online compendium to the article: An analysis of heuristics for vertex colouring," https://imada.sdu.dk/~marco/gcp-study/, accessed: 2020-01-10.

[71] J. Culberson. Graph coloring page. [Online]. Available: https://webdocs.cs.ualberta.ca/~joe/Coloring/

[72] D. de Werra, "Some models of graphs for scheduling sports competitions," *Discrete Applied Mathematics*, vol. 21, no. 1, pp. 47 – 65, 1988. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X88900339

[73] R. Lewis and J. Thompson, "On the application of graph colouring techniques in round-robin sports scheduling," *Comput. Oper. Res.*, vol. 38, no. 1, p. 190–204, Jan. 2011. [Online]. Available: https://doi.org/10.1016/j.cor.2010.04.012

[74] "Graph coloring resources benchmark instances for (distributed) graph coloring," https://www.cs.upc.edu/~hhernandez/graphcoloring/, accessed: 2020-04-10.

[75] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, "Towards objective measures of algorithm performance across instance space," *Computers & Operations Research*, vol. 45, p. 12–24, 05 2014.

[76] D. Corne and A. Reynolds, "Optimisation and generalisation: Footprints in instance space," 09 2010, pp. 22–31.

[77] N. Musliu and M. Schwengerer, "Algorithm selection for the graph coloring problem," *Revised Selected Papers of the 7th International Conference on Learning and Intelligent Optimization - Volume 7997*, p. 389–403, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-44973-4_42

[78] K. Smith-Miles and D. Baatar, "Exploring the role of graph spectra in graph coloring algorithm performance," *Discrete Applied Mathematics*, vol. 176, p. 107–121, 10 2014.

[79] M. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, pp. 167–256, 2003.

[80] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1038/30918

[81] L. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, pp. 35–41, 03 1977.

[82] P. Hage and F. Harary, "Eccentricity and centrality in networks," *Social Networks*, vol. 17, no. 1, pp. 57 – 63, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0378873394002489

[83] R. Balakrishnan, "The energy of a graph," *Linear Algebra and its Applications*, vol. 387, pp. 287 – 295, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0024379504001259

[84] B. Mohar, Y. Alavi, G. Chartrand, O. Oellermann, and A. Schwenk, "The laplacian spectrum of graphs," *Graph Theory, Combinatorics and Applications*, vol. 2, p. 5364, 01 1991.

[85] K. Smith-Miles, B. Wreford, L. Lopes, and N. Insani, "Predicting metaheuristic performance on graph coloring problems using data mining," *Studies in Computational Intelligence*, vol. 434, pp. 417–432, 01 2013.

[86] N. Insani, K. Smith-Miles, and D. Baatar, "Selecting suitable solution strategies for classes of graph coloring instances using data mining," in *2013 International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2013, pp. 208–215.

[87] T. Mehrotra, "A column generation approach for graph coloring," *INFORMS J. on Computing*, vol. 8, no. 4, p. 344–354, Nov. 1996. [Online]. Available: https://doi.org/10.1287/ijoc.8.4.344

[88] M. Chiarandini, T. Stützle *et al.*, "An application of iterated local search to graph coloring problem," in *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, 2002, pp. 112–125.

[89] X.-F. Xie and J. Liu, "Graph coloring by multiagent fusion search," *Journal of combinatorial optimization*, vol. 18, no. 2, pp. 99–123, 2009.

[90] E. Malaguti, M. Monaci, and P. Toth, "A metaheuristic approach for the vertex coloring problem," *INFORMS Journal on Computing*, vol. 20, no. 2, pp. 302–316, 2008.

[91] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artificial Intelligence Review*, vol. DOI: 10.1007/s10462-013-9406-y, 06 2013.

[92] S. Ali and K. Smith-Miles, "On learning algorithm selection for classification," *Applied Soft Computing*, vol. 6, pp. 119–138, 01 2006.

[93] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: The state of the art," *CoRR*, vol. abs/1211.0906, 2012. [Online]. Available: http://arxiv.org/abs/1211.0906

[94] T. Kohonen, *Self-Organized Formation of Topologically Correct Feature Maps*. Cambridge, MA, USA: MIT Press, 1988, p. 509–521.

[95] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[96] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[97] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017.

[98] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 6638–6648. [Online]. Available: http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf

[99] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, Feb. 2012.

[100] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523.

[101] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13. JMLR.org, 2013, p. I–115–I–123.

[102] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, no. 1–3, p. 389–422, Mar. 2002. [Online]. Available: https://doi.org/10.1023/A:1012487302797

[103] I. Huerta, D. Neira, D. Ortega, V. Varas, J. Godoy, and R. Asín-Achá, "Anytime automatic algorithm selection for knapsack," *Expert Systems with Applications*, p. 113613, 06 2020.

[104] "Graph coloring benchmarks," https://sites.google.com/site/graphcoloring/files, accessed: 2020-01-10.

[105] "Benchmark data for the paper:new integer linear programming models for the vertex coloring problem," https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks, accessed: 2020-01-10.

[106] "sports scheduling benchmark problem files," http://www.rhydlewis.eu/resources/PrincipalityPremProbs.zip, accessed: 2020-01-10.

[107] R. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *J. Heuristics*, vol. 7, pp. 261–304, 05 2001.

[108] B. Schling, *The Boost C++ Libraries*. XML Press, 2011.

[109] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014. [Online]. Available: http://figshare.com/articles/graph_tool/1164194

[110] M. A. Jette, A. B. Yoo, and M. Grondona, "Slurm: Simple linux utility for resource management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[111] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Roy. Stat. Soc.*, vol. 36, pp. 111–147, 1974.

[112] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[113] C. Chen and L. Breiman, "Using random forest to learn imbalanced data," *University of California, Berkeley*, 01 2004.

[114] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[115] G. Louppe and P. Geurts, "Ensembles on random patches," 09 2012, pp. 346–361.

[116] Q. Zhu, "On the performance of matthews correlation coefficient (mcc) for imbalanced dataset," *Pattern Recognition Letters*, vol. 136, pp. 71 – 80, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786552030115X

[117] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS'11. Red Hook, NY, USA: Curran Associates Inc., 2011, p. 2546–2554.

[118] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," 2018.

A. Anexos

A.1. Parámetros utilizados en la generación de instancias

Para *Joe Cuberlson* se usaron 3 esquemas disponibles:

- **Coloreos equipartitos:** Con esta configuración se generaron 1000 grafos del tipo *Independant Random Edge Assignment* usando los siguientes parámetros: La cantidad de nodos n : 150, 250, 400, 750; la probabilidad de asignar una arista entre vértices p : 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95 y la semilla para el generador aleatorio s : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25.
- **K-coloring:** Con esta configuración se generaron 1000 grafos del tipo *Independant Random Edge Assignment* usando como parámetros: La cantidad de nodos n : 150, 250, 400, 750, la probabilidad de asignar una arista entre vértices p : 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95 y la semilla para el generador aleatorio s : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, el número de k particiones que corresponde a $n * p$ y la variabilidad dado por $\lfloor (k-1) * p + 1 \rfloor$
- **No-hidden:** Con esta configuración se generaron 1000 grafos del tipo *Independant Random Edge Assignment* y tiene la misma configuración que coloreos equipartitios.

Para *Networkx* según el tipo de grafo se usaron los siguientes parámetros:

- **Watts strogatz, Newman watts strogatz, Connected watts strogatz:** Para cada uno de los tipos de grafos se generaron 210 grafos, usando los parámetros: n : 250, 500, 750, 1000, 1250, 1500, 1750, 2000; k :100, 200, 500, 50, 750 y $prob$:0.1, 0.3, 0.5, 0.7, 0.9, 1. En este tipo de grafo, $prob$ corresponde a la probabilidad de reconexión de cada arco.
- **Duplication divergence:** Se generaron 84 grafos, usando los parámetros n : 250, 500, 750, 1000, $prob$:0.1, 0.3, 0.5, 0.7, 0.9, 1 y q : 0.1, 0.3, 0.5, 0.7, 0.9, 1. Estos grafos son inspirados en procesos biológicos.

- **Partial duplication graph:** Se generaron 84 grafos, usando los parámetros: N : 300, 600, 800, 1050, 1200, n : 250, 500, 750, 1000, $prob$:0.1, 0.3, 0.5, 0.7, 0.9, 1 y q :0.1, 0.3, 0.5, 0.7, 0.9, 1.
- **Gnm random y Dense gnm random:** Se generaron 256 grafos de cada tipo usando usando los parámetros: n : 250, 500, 750, 1000, 1250, 1500, 1750, 2000. m : 150, 300, 400, 450, 525, 600, 750, 900, 1050, 1200, 1500, 1575, 1600, 1800, 2000, 2100, 2400, 2625, 2800, 3000, 3150, 3200, 3600, 3675, 4200, 4800.
- **Random Regular:** Se generaron 39 grafos, usando los parámetros: d :50, 100, 200, 500, 750, 1000 y n :250, 500, 750, 1000, 1250, 1500, 1750, 2000.

Para *Hisampler* se uso el solver DSATUR como solver base para la generación de instancias, utilizando los parámetros por defectos para el entrenamiento, excepto por el número de nodos n: 50, 100, 150, 300, 500, 750, 1000, 1500. También se realizaron pruebas usando como base el solver *Head*, esto principalmente para tratar de evitar el desbalanceo que se genera en los datos (mencionados más adelante); sin embargo, las pruebas no fueron efectivas ya que las instancias generadas fueron difíciles para todos los solvers. Del conjunto de instancias totales se eliminaron 572 instancias no conexas, por lo que se trabajaron solo con 6693.

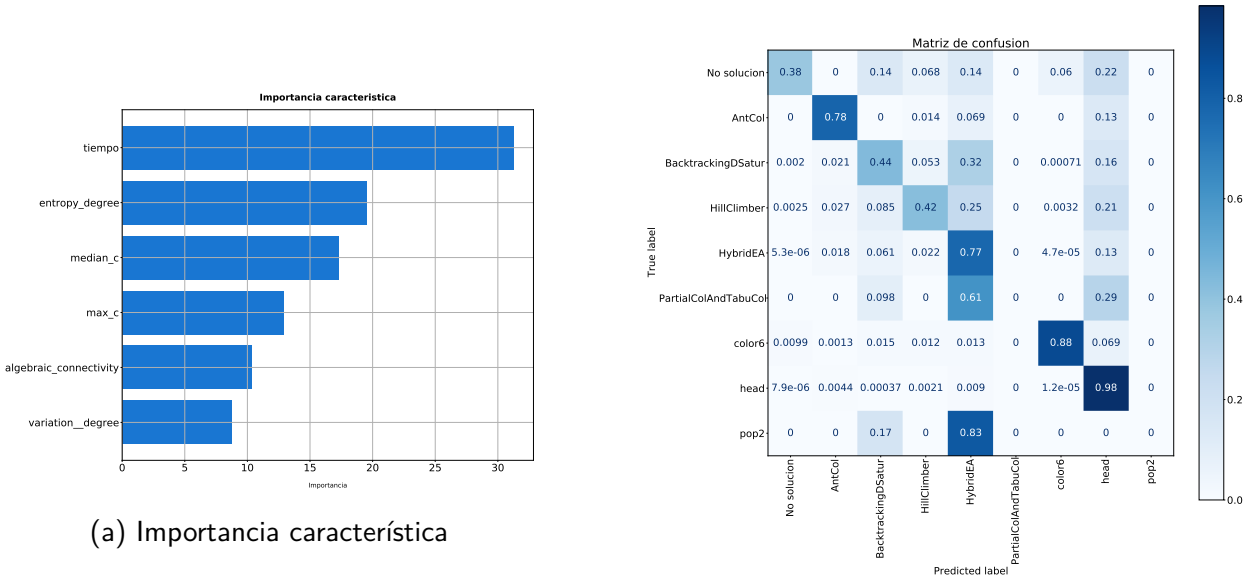
A.2. Descripción de los datos

Se describen los archivos encontrados en `https://drive.google.com/drive/folders/1qJcBoPNiX3fX3lqVEjEpGBszFliEktE8?usp=sharing`. Estos corresponde a los datos base, para la realización de los experimentos descritos en este trabajo.

- **clean_instances.zip** : Corresponde a las instancias recolectadas.
- **Solvers**: En esta carpeta se encuentran los códigos fuentes de los solvers modificado, para seguir una ejecución de algoritmos anytime.
- **Features**: En esta carpeta se encuentra el código fuente para realizar el calculo de características sobre las instancias.

- **clasificacion-lineal.pkl**: Corresponde a los tiempos utilizados para entrenar el enfoque de clasificación, seleccionando el solver ganador por tiempo, usando la curva de tiempo lineal y todos los solvers disponibles.
- **Regresión times**: En esta carpeta se encuentran tiempos utilizados para el modelo de regresión.
- **tiempos-lineales-regresion**: En esta carpeta se encuentran los tiempos generados para cada solver.
- **tiempos-limpios.csv**: Corresponde a los tiempos obtenidos de la ejecución de los algoritmos, hay que hacer una limpieza de datos.
- **feats-limpios.csv**: Corresponde a los valores obtenidos del calculo de las características.
- **Clasification model:** Contiene un jupyter notebook, que permite ejecutar el modelo de clasificación entrenado, reportando las métricas y gráficos reportados asociados en clasificación en este trabajo.

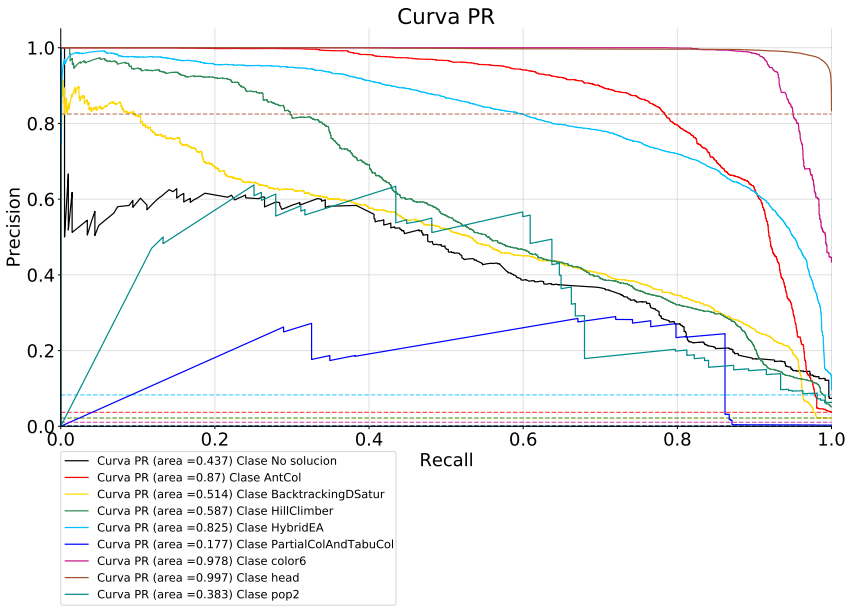
A.3. Mejor modelo clasificación usando MCC



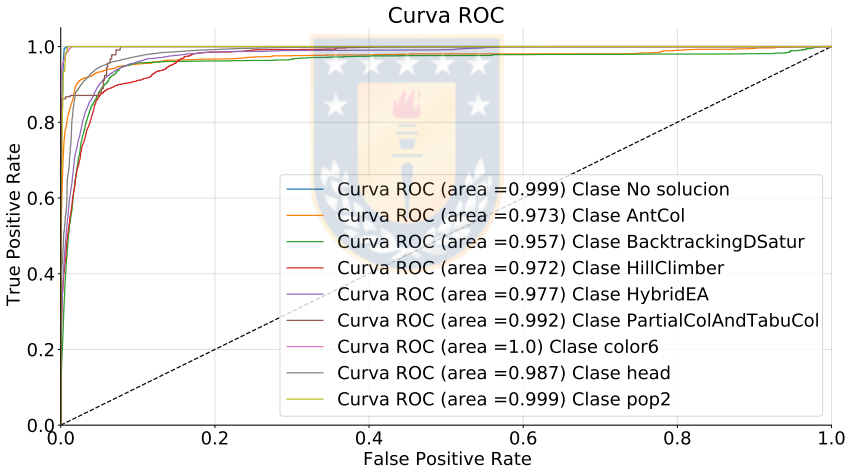
(a) Importancia característica

(b) Matriz de confusión

Fig. A.1: Gráficos clasificación usando MCC como métrica



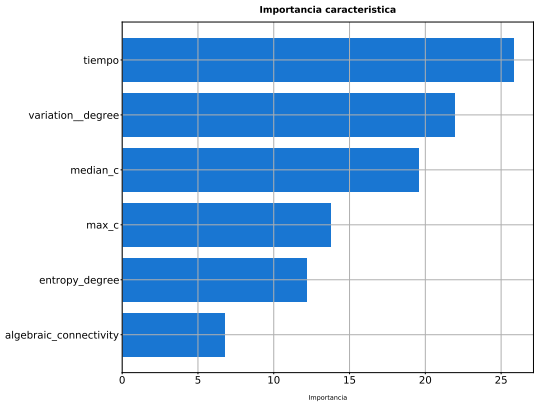
(a) Curva precision-recall



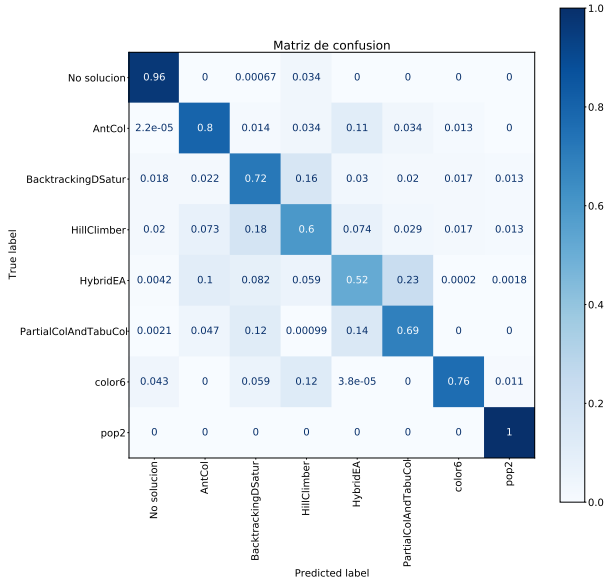
(b) Curva ROC

Fig. A.2: Curva ROC - PR usando GM como métrica

A.4. Mejor modelo clasificación sin Head

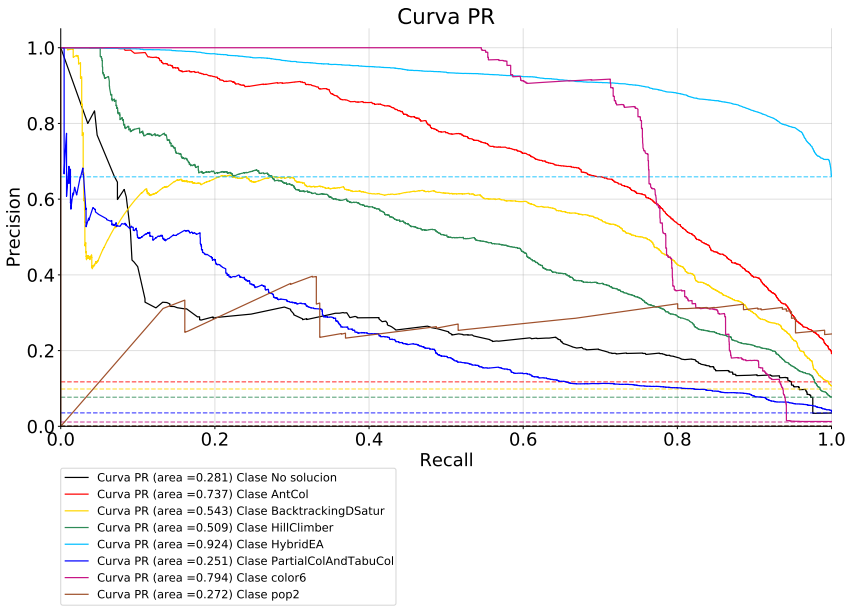


(a) Importancia característica sin Head

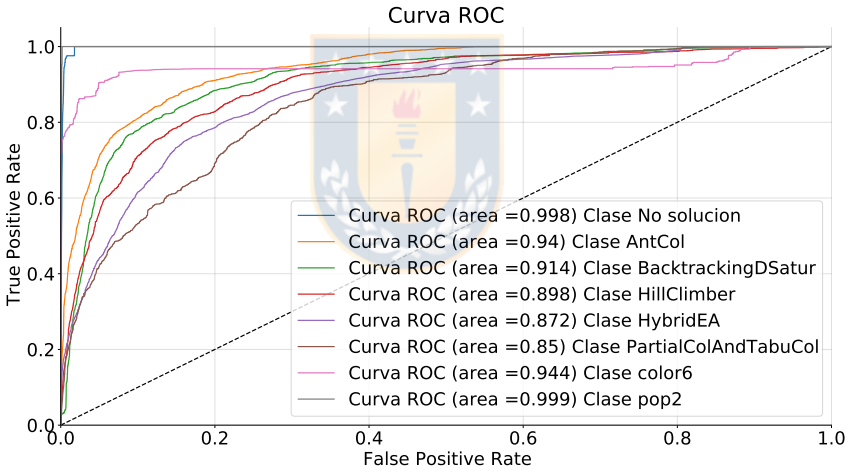


(b) Matriz de confusión sin Head

Fig. A.3: Gráficos clasificación sin Head usando GM como métrica



(a) Curva precision-recall sin Head



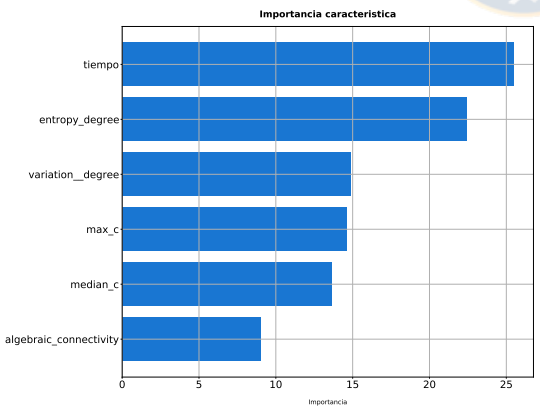
(b) Curva ROC sin Head

Fig. A.4: Curva ROC - PR sin Head usando GM como métrica

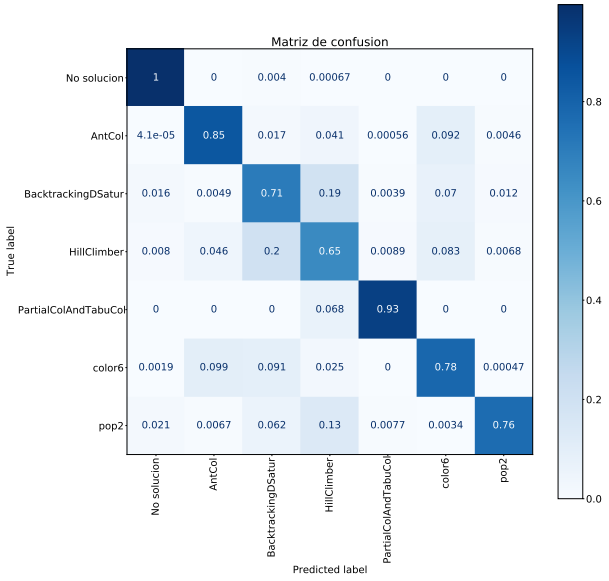
Metrica	Valor
accuracy	0.586
recall_weighted	0.586
f1_weighted	0.626
precision_weighted	0.77
cohen_kappa_score	0.416
matthews_corrcoef	0.46
gm	0.74
gm_macro	0.842
gm_micro	0.742
gm_weight	0.734

Tabla A.1: Métricas mejor clasificador sin Head sobre conjunto de test

A.5. Mejor modelo clasificación sin Head y HybridEA

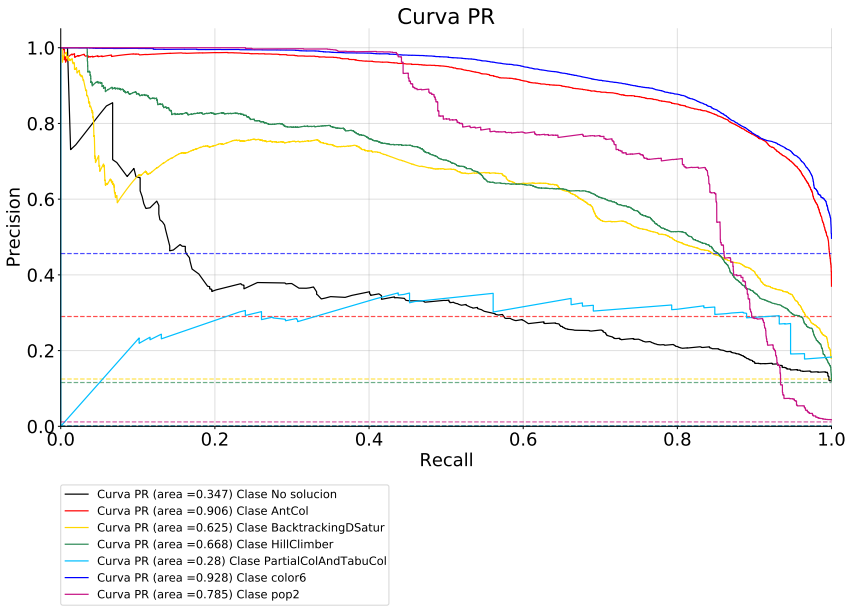


(a) Importancia característica sin Head y HybridEA

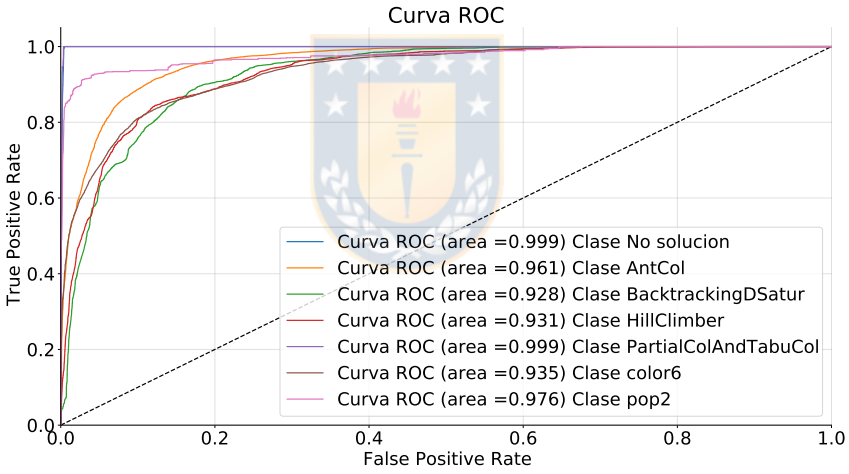


(b) Matriz de confusión sin Heady HybridEA

Fig. A.5: Gráficos clasificación sin Head y HybridEA usando GM como métrica



(a) Curva precision-recall sin Head y HybridEA



(b) Curva ROC sin Head y HybridEA

Fig. A.6: Curva ROC - PR sin Head y HybridEA usando GM como métrica

Metrica	Valor
accuracy	0.776
recall_weighted	0.776
f1_weighted	0.782
precision_weighted	0.794
cohen_kappa_score	0.678
matthews_corrcoef	0.68
gm	0.803
gm_macro	0.881
gm_micro	0.864
gm_weight	0.847

Tabla A.2: Métricas mejor clasificador sin Head y sin HybridEA sobre conjunto de test

A.6. Tablas Comparación

Timeout (ms)	6F Sin considerar tiempo	6F Considera tiempo	6F Diferencia	20F Sin considerar tiempo	20F Considera tiempo	20F Diferencia	Head sin tiempo
540	0.886	0.452	0.435	0.918	0.426	0.492	0.824
2400	0.900	0.619	0.281	0.921	0.597	0.324	0.874
3000	0.893	0.643	0.250	0.914	0.639	0.275	0.884
6000	0.895	0.713	0.182	0.903	0.707	0.196	0.895
7200	0.900	0.728	0.171	0.906	0.720	0.186	0.900
9000	0.895	0.742	0.153	0.900	0.737	0.162	0.905
10800	0.887	0.746	0.141	0.892	0.744	0.149	0.910
12000	0.888	0.750	0.138	0.895	0.747	0.148	0.916
24000	0.878	0.793	0.085	0.920	0.803	0.117	0.930
60000	0.902	0.846	0.056	0.936	0.870	0.067	0.940
900000	0.938	0.937	0.001	0.952	0.950	0.001	0.943
1500000	0.938	0.938	0.000	0.952	0.952	0.000	0.943
1800000	0.938	0.938	0.000	0.952	0.952	0.000	0.943
2640000	0.938	0.938	0.000	0.952	0.952	0.000	0.943

Tabla A.3: Proporción de acierto solución comparando modelos con 6 y 20 características.

Timeout (ms)	GM considerar tiempo	GM Considera tiempo	GM Diferencia	MCC Sin considerar tiempo	MCC Considera tiempo	MCC Diferencia	Head sin tiempo
540	0.886	0.452	0.435	0.919	0.457	0.461	0.824
2400	0.900	0.619	0.281	0.935	0.623	0.312	0.874
3000	0.893	0.643	0.250	0.927	0.648	0.279	0.884
6000	0.895	0.713	0.182	0.929	0.724	0.205	0.895
7200	0.900	0.728	0.171	0.939	0.735	0.204	0.900
9000	0.895	0.742	0.153	0.933	0.754	0.179	0.905
10800	0.887	0.746	0.141	0.932	0.761	0.171	0.910
12000	0.888	0.750	0.138	0.931	0.769	0.163	0.916
24000	0.878	0.793	0.085	0.954	0.827	0.127	0.930
60000	0.902	0.846	0.056	0.954	0.896	0.058	0.940
900000	0.938	0.937	0.001	0.967	0.966	0.001	0.943
1500000	0.938	0.938	0.000	0.967	0.967	0.000	0.943
1800000	0.938	0.938	0.000	0.967	0.967	0.000	0.943
2640000	0.938	0.938	0.000	0.967	0.967	0.000	0.943

Tabla A.4: Proporción de acierto solución comparando métricas GM y MCC.