UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Inteligencia Computacional para la Generación de Modelos Subrogados

Informe de Proyecto de Grado presentado por

Miguelángel Díaz, Rodrigo Gordienko

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Martín Pedemonte
Jimena Ferreira

Montevideo, 20 de diciembre de 2024

Agradecimientos

En esta etapa final de nuestras carreras de grado, queremos expresar nuestro más profundo agradecimiento a todas las personas que, de una u otra forma, hicieron posible la realización de este proyecto.

En primer lugar, extendemos nuestro más sincero agradecimiento a Martín Pedemonte y Jimena Ferreira, quienes nos guiaron con dedicación, paciencia y sabiduría a lo largo de este camino. Su incansable orientación, apoyo y experiencia en cada etapa del trabajo fueron esenciales para el éxito de este proyecto.

Asimismo, reconocemos el papel crucial de la Facultad de Ingeniería y sus distinguidos profesores. Su compromiso con la excelencia académica y su valiosa enseñanza constituyeron la base sobre la cual construimos este proyecto. La educación recibida en esta institución no solo nos brindó las herramientas necesarias, sino que también nos inspiró a buscar siempre la mejora continua en nuestra formación profesional.

Por último, pero no menos importante, agradecemos a nuestras familias, quienes con su amor incondicional, paciencia y apoyo fueron nuestra fuente constante de inspiración y fortaleza. A nuestros amigos, por su aliento permanente, por escuchar con empatía y por brindarnos palabras de ánimo en los momentos más desafiantes.

A todos ustedes, nuestro más sincero ¡gracias!

Resumen

Los modelos subrogados son aproximaciones que buscan replicar el comportamiento de modelos numéricos o simulaciones de manera precisa, utilizando ecuaciones simples formadas por combinaciones de polinomios, funciones logarítmicas, exponenciales, entre otras. Este trabajo tiene como objetivo explorar y aplicar técnicas de inteligencia computacional para la generación de modelos subrogados a partir de datos de simulación, abordando tanto problemas de *benchmarking* como casos reales en Ingeniería de Procesos.

Para ello, se evaluaron tres enfoques principales: DSO, Operon y DSR. En una primera etapa, se realizó una experimentación monoobjetivo, utilizando el RMSE como métrica principal para comparar su desempeño y aplicando métodos estadísticos como el Test de Friedman y el procedimiento *post-hoc* de Holm. Posteriormente, se extendió DSO para abordar problemas de optimización multiobjetivo, incorporando objetivos como la minimización de puntos indefinidos, el cumplimiento de restricciones de monotonía y la precisión. Finalmente, se desarrolló una variante de DSO con un esquema de selección mejorado, que utiliza el operador de selección de Deb, enfocado en priorizar restricciones relacionadas con las propiedades estructurales de las soluciones.

Los resultados obtenidos evidenciaron que Operon destacó en la experimentación monoobjetivo, logrando menores RMSE y mayor consistencia en comparación con DSO y DSR, según el análisis estadístico realizado. Con respecto al enfoque multiobjetivo, aunque la versión multiobjetivo de DSO implementada mostró potencial para manejar restricciones como monotonía y minimizar puntos indefinidos en algunos problemas de lo *benchmarks* evaluados, su desempeño fue inconsistente para otros, con alta variabilidad, mayores costos computacionales y menor cumplimiento de las propiedades físicas en algunos casos. Sin embargo, el operador de selección de Deb demostró ser altamente efectivo para satisfacer propiedades físicas como la monotonía, y reducir la cantidad de puntos indefinidos respecto a los anteriores enfoques.

Palabras clave: Modelos Subrogados, Regresión Simbólica, Programación Genética, Ingeniería de Procesos, Redes Neuronales, Modelos Híbridos, Machine Learning

Índice general

Capítulo 1

Introducción

1.1. Motivación

En el último siglo, un área fundamental de las ciencias y tecnologías ha sido la creación de modelos que describen y explican comportamientos de la física y/o química, a partir de datos experimentales y leyes de la física, bajo determinadas condiciones. Estos modelos han sido muy importantes para el avance tecnológico en una amplia variedad de industrias como la ingeniería de procesos químicos.

Con la llegada de la inteligencia artificial (IA) y los avances en modelos de aprendizaje automático (ML, según su sigla en inglés) y aprendizaje profundo (DL, según su sigla en inglés), es natural pensar que estos enfoques podrían complementar o sustituir las simulaciones tradicionales. Sin embargo, muchos de los modelos actuales se comportan como *cajas negras*, donde la estructura o forma de la función es desconocida o demasiado compleja para ser comprendida por un ser humano. Esta naturaleza opaca, a menudo intencional, es efectiva para proteger la propiedad intelectual y ha motivado el desarrollo de modelos aún más sofisticados, dificultando su interpretación y favoreciendo la creación de sistemas más difíciles de replicar mediante ingeniería inversa.

Si bien los modelos de *caja negra* pueden proteger los intereses comerciales, su opacidad plantea serios desafíos en contextos donde las decisiones basadas en ellos implican riesgos significativos. En estos escenarios, la capacidad de explicar cómo un modelo toma decisiones es crítica, ya que no solo ayuda a mitigar el riesgo, sino que también refuerza la confianza en las soluciones que se implementan. Asimismo, intentar crear modelos explicativos que interpreten las *cajas negras* suele ser problemático, ya que estos tienden a no capturar completamente el comportamiento del modelo original. Además, las explicaciones resultantes pueden ser tan complejas que, paradójicamente, se vuelven tan incomprensibles como el modelo que intentan interpretar.

La complejidad de un modelo tampoco garantiza una mayor precisión, especialmente en problemas con datos estructurados y *features* significativas. Además, los modelos de *caja negra* a menudo requieren que toda la información se cen-

tralice en un formato específico, lo que no siempre es factible en entornos con decisiones críticas y dispersión de datos. Por lo tanto, varios autores sugieren el desarrollo de modelos interpretativos que no solo ofrezcan soluciones precisas, sino que también se adapten mejor a la realidad de cada contexto, ofreciendo mayor transparencia y confianza en el proceso de toma de decisiones (Rudin, 2019).

A los modelos que replican la relación entrada-salida de un sistema, garantizando tanto la transparencia en la inferencia como una complejidad razonable, se les denomina modelos subrogados. En este contexto, la regresión simbólica ha surgido como una técnica que, a diferencia de los modelos tradicionales de ML o DL, busca identificar una función simbólica que no solo se ajuste a los datos, sino que también sea comprensible para los humanos.

1.2. Objetivos

El objetivo general de este proyecto es estudiar la generación de modelos subrogados utilizando técnicas de inteligencia computacional, en particular algoritmos evolutivos mediante programación genética y redes neuronales para regresión simbólica, con aplicación a problemas relevantes en ingeniería de procesos. Para ello, se realizará un relevamiento exhaustivo de algoritmos y métodos prometedores, los cuales serán evaluados y validados tanto en problemas de *benchmarking* como en casos reales.

Por otra parte, dado que en general, los algoritmos clásicos suelen tener inconvenientes para generar modelos que respeten las condiciones impuestas por la física o química en problemas reales, se estudiará el diseño de variantes ajustadas a partir de estos métodos validados, adaptándolos para su aplicación en el contexto específico de las problemáticas abordadas.

1.3. Estructura

En el Capítulo 2 se presenta el marco teórico, donde se abordan los conceptos fundamentales que sustentan este trabajo, tales como los modelos subrogados, la regresión simbólica (RS), los algoritmos evolutivos (AEs), la programación genética (PG), las redes neuronales aplicadas a la RS y los algoritmos híbridos como *Deep Symbolic Optimization* (DSO). Estos conceptos constituyen la base para entender los enfoques y las metodologías adoptadas en el desarrollo de este proyecto.

Además, en este capítulo se realiza un análisis de los algoritmos evolutivos multiobjetivo, haciendo énfasis en su aplicación para la optimización de modelos simbólicos. Se examinan herramientas como jMetalPy, que facilitan la implementación de estos algoritmos en problemas complejos. También se abordan los avances recientes en DL, especialmente en el uso de redes neuronales profundas para mejorar la generación de modelos simbólicos, lo cual incluye la integración de técnicas de aprendizaje profundo como las redes neuronales

recurrentes (RNN), que pueden generar expresiones simbólicas más precisas y estructuradas en el contexto de la regresión simbólica.

En el Capítulo 3, se examinan las soluciones propuestas en trabajos relacionados, con especial atención a problemas relevantes, como los abarcados por los *benchmarks* de Feynman y Vladislavleva (presentados posteriormente en las secciones 4.4.2 y 4.4.3). Esta revisión de antecedentes permite identificar las fortalezas y debilidades de enfoques previos y sirve como punto de partida para las mejoras desarrolladas en este proyecto.

Posteriormente, se seleccionaron tres trabajos clave que sirven como punto de partida. DSO, explicado en la Sección 2.5 que aborda la generación de modelos mediante un algoritmo híbrido de ML y PG, que utiliza una RNN para generar la población inicial de cada iteración de un algoritmo de PG. También *Deep Symbolic Regression*, explicado en la Sección 2.4.4, que abarca únicamente el componente de ML del anterior *framework*. Por otro lado, Operon, un algoritmo de PG basado en técnicas de mínimos cuadrados para la evaluación de los individuos explicado en la Sección 2.3.2 de este documento.

En el Capítulo 4, se presenta la metodología seguida durante la experimentación, se proporcionan detalles sobre los *frameworks*, su instalación, las modificaciones realizadas para facilitar el relevamiento de resultados y los cambios realizados para adaptar los algoritmos a los experimentos. A su vez, se detallan los *benchmarks* utilizados.

En la fase de experimentación de este proyecto, en primer lugar se llevó a cabo un análisis de resultados con enfoque monoobjetivo empleando los algoritmos seleccionados, cuyo desempeño se evaluó utilizando la raíz del error cuadrático medio (RMSE, según su sigla en inglés) como métrica principal. Para analizar dichos resultados, se aplicaron métodos estadísticos como el Test de Friedman y el procedimiento *post-hoc* de Holm, presentando los resultados mediante medidas descriptivas.

Posteriormente, se implementó una extensión de DSO diseñada para abordar problemas de optimización multiobjetivo. En esta versión, los objetivos a minimizar incluyen, además del RMSE, la cantidad de puntos donde la función es indefinida e indicadores relacionados con el cumplimiento de restricciones de monotonía.

Finalmente, se desarrolló una tercera variante del algoritmo DSO, que modifica la selección para priorizar de manera más estricta el cumplimiento de las restricciones relacionadas con la forma de las soluciones, como las mencionadas anteriormente.

Estos experimentos se documentan en los Capítulos 5, 6 y 7, respectivamente. En cada capítulo se describen las configuraciones realizadas para llevar a cabo las pruebas, los resultados obtenidos y su análisis. Asimismo, los resultados se presentan a través de tablas y gráficos, comparando los resultados obtenidos para distintos *benchmarks*, de las técnicas propuestas, para evaluar su desempeño.

El informe concluye en el Capítulo 8, donde se exponen las conclusiones alcanzadas a partir de los resultados obtenidos, junto con propuestas para futuras líneas de investigación que podrían mejorar o extender los logros alcanzados.

Capítulo 2

Marco Teórico

En este capítulo se abordan los modelos subrogados, la regresión simbólica, los algoritmos evolutivos, junto con las redes neuronales aplicadas a la regresión simbólica y los modelos híbridos como *Deep Symbolic Optimization*. Estos conceptos son esenciales dentro del marco teórico de la inteligencia computacional, proporcionando una base sólida para comprender la integración de técnicas avanzadas de optimización y aprendizaje automático en la generación de modelos subrogados. Se analiza cómo la sinergia entre estos enfoques permite abordar problemas complejos y reales de manera eficiente.

2.1. Modelos Subrogados

Los modelos subrogados son una alternativa en áreas como la Ingeniería mecánica y química, donde modelos rigurosos son comunes para modelar fenómenos físicos y/o químicos de sistemas complejos, generando grandes sistemas de ecuaciones algebraicas, diferenciales o diferenciales-algebraicas. La resolución de estos sistemas, es costosa computacionalmente y puede tardar mucho tiempo en realizarse. Esto dificulta tareas como la optimización del diseño, la exploración del espacio de soluciones y el análisis de sensibilidad.

Para superar esta limitación, se utilizan modelos subrogados, que son modelos aproximados que imitan el comportamiento de las simulaciones numéricas de manera más rápida y eficiente. Estos modelos se construyen a partir de un enfoque basado en datos, donde se modela la respuesta del simulador a un número limitado de puntos de datos seleccionados de manera inteligente. De esta forma, los modelos subrogados en general logran capturar suficientemente el comportamiento de entrada-salida para predecir el funcionamiento de un fenómeno frente a diversas condiciones, con miras a su optimización.

Los modelos subrogados suelen ser clasificados como modelos de caja negra debido a que el funcionamiento del sistema se observa únicamente en función de sus entradas y salidas, sin que se disponga de ningún conocimiento interno adicional sobre su operación.

Una lista de los métodos más comunes para producir modelos a partir de datos es la siguiente: regresión lineal (Box y Draper, 1986), regresión no lineal (Johnson y Wichern, 1988), incluyendo interpolaciones racionales multivariantes en puntos (Cuyt y Verdonk, 1985) y en segmentos verticales (Celis, Cuyt, y Verdonk, 2007), aproximaciones multipunto de superficies de respuesta para optimización (Toropov, 1989), mínimos cuadrados móviles (Choi, Youn, y Yang, 2002; Levin, 1998), *Kriging* (también conocido como *Gaussian Process*) (Kleijnen, 2008; Sacks, Welch, Mitchell, y Wynn, 1989), splines de regresión adaptativa multivariante (Friedman, 1991), funciones de base radial (*radial basis functions*) (M. J. D. Powell, 1987), redes neuronales (Haykin, 1994), regresión de vectores de soporte (*support vector regression*) (Cherkassky y Mulier, 1998; Vapnik, 1997), programación genética (Koza, 1994), entre otros.

2.2. Regresión simbólica

La IA interpretativa se enfoca en diseñar modelos que puedan ser entendidos por expertos. Un aspecto clave de este enfoque es la regresión simbólica (RS), que busca encontrar una función simbólica que pueda aproximar un conjunto de datos finito. La RS proporciona un método para identificar funciones a partir de los datos, enfrentando el desafío de encontrar expresiones que se ajusten bien al conjunto de entrenamiento y que también generalicen correctamente con nuevos datos.

La teoría subyacente en la RS parte de la premisa de que existe una función generadora de los datos, la cual puede ser recuperada mediante métodos de RS para encontrar una expresión que transforme de manera adecuada las entradas del modelo en la salida deseada. Esta expresión debería ser capaz de generalizar los resultados incluso más allá del conjunto de entrenamiento.

Un principio fundamental en la RS es la parsimonia, también conocida formalmente como el principio de parsimonia, aunque a veces se le cita anecdóticamente como la navaja de Occam (Bruneton, Cazenille, Douin, y Reverdy, 2019). Este principio postula que el modelo más simple que pueda explicar el fenómeno observado es preferible. En la práctica, esto significa que el objetivo es desarrollar un modelo útil para describir el fenómeno en cuestión, utilizando un número manejable de variables que funcionen como buenos predictores. De este modo, se evita la complejidad excesiva que podría llevar a un modelo que no sea práctico (Burlacu, Kronberger, Kommenda, y Affenzeller, 2019).

La parsimonia implica que un modelo que cumple con este principio es aquel que logra el nivel deseado de explicación con el menor número de variables. Esto no solo facilita la interpretación y el análisis del modelo, sino que también ayuda a prevenir el sobreajuste y asegura que el modelo sea lo suficientemente robusto para generalizar a nuevos datos (Derner, Kubalík, Ancona, y Babuška, 2020).

Los problemas en RS se dividen en dos partes: el problema de ajuste de parámetros, relacionado con la optimización de coeficientes, y el problema estructural, relacionado con la estructura de la solución. Los modelos en la literatura abordan estos problemas con diferentes enfoques, a menudo poniendo un

mayor énfasis en uno de ellos.

En el contexto de sistemas dinámicos y procesos en áreas críticas como la ingeniería de control, es esencial contar con modelos confiables, robustos y seguros que se ajusten estrictamente a las expectativas de comportamiento según el conocimiento específico del dominio, así como a criterios de seguridad y rendimiento. Sin embargo, tales modelos no siempre pueden derivarse únicamente de principios fundamentales y consideraciones teóricas. Por lo tanto, deben ser determinados empíricamente (Kronberger, de Franca, Burlacu, Haider, y Kommenda, 2021).

Desde esta perspectiva, los modelos utilizados en la identificación de sistemas pueden clasificarse en tres categorías:

- **Modelos de caja blanca (*White-box*):** Son aquellos modelos derivados de principios fundamentales o leyes físicas, con una estructura completamente interpretable por diseño. Permiten una comprensión clara del proceso y una validación a través del conocimiento del dominio, lo que facilita su aplicación a un amplio rango de entradas, incluso más allá de las observaciones disponibles. Sin embargo, suelen ser menos precisos que otros enfoques más complejos. También pueden tener una mayor complejidad computacional. Por ejemplo, si implican la resolución de ecuaciones en derivadas parciales (Rudin, 2019).

- **Modelos de caja negra (*Black-box*):** Son modelos altamente complejos, como las redes neuronales profundas (DNN, según su sigla en inglés), cuyo funcionamiento interno es difícil o incluso imposible de inspeccionar directamente. Aunque estos modelos suelen ser más precisos, su opacidad plantea problemas de confianza y comprensión por parte de los usuarios. Las DNN, en particular, dependen de numerosos factores en su diseño estructural, como las funciones de activación, el tamaño y tipo de entradas, el número de capas, las operaciones de agrupamiento, y los mecanismos de actualización de pesos, lo que contribuye a su naturaleza de *black-box* (Ali, Abuhmed, El-Sappagh, y Muhammad, 2023).

- **Modelos de caja gris (*Gray-box*):** Entre los extremos de los modelos de caja negra y caja blanca, existe todo un espectro de modelos de caja gris que combinan aspectos de ambos enfoques. Por un lado, pueden incorporar una estructura basada en conocimientos previos, lo que proporciona cierto grado de interpretabilidad. Por otro lado, también dependen de los datos para ajustar parámetros específicos, lo que les permite una mayor precisión en problemas específicos, sin llegar a ser completamente opacos. Ejemplos de este tipo de modelos incluyen sistemas basados en reglas difusas (FRBS, según su sigla en inglés) o redes bayesianas, los cuales ofrecen un equilibrio razonable entre interpretabilidad y precisión (Ali y cols., 2023; Ljung, 2010).

Los defensores de la *IA explicativa* proponen aplicar modelos en el extremo blanco de este espectro, ya sea directamente (Rudin, 2019) o como una aproximación de modelos de caja negra en tecnologías de IA. En la Figura 2.1 se

puede apreciar las mencionadas diferencias entre los distintos enfoques de forma más visual.

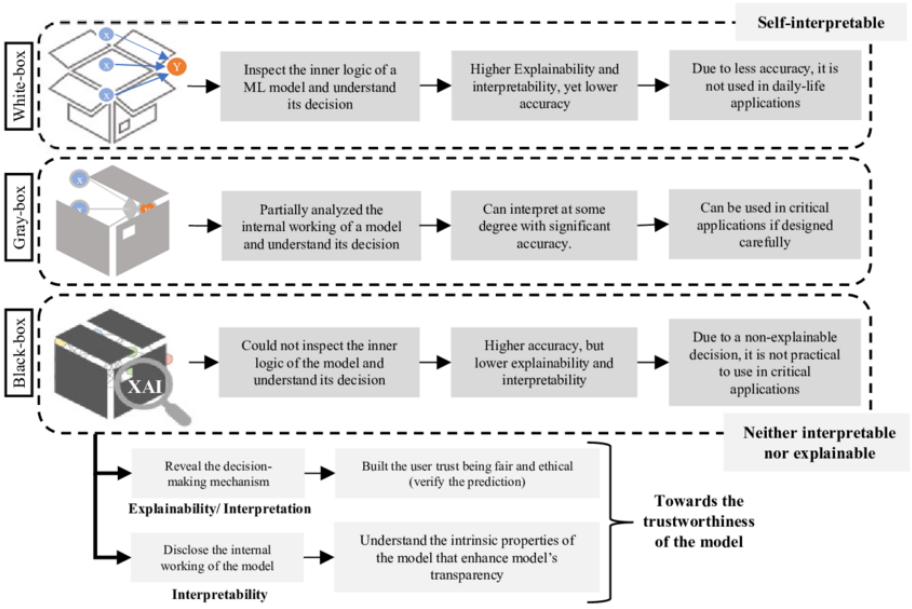

Figura 2.1: Comparación entre modelos caja blanca, gris y negra. Imagen extraída de Ali y cols. (2023).

La RS resulta particularmente interesante en este contexto, ya que los modelos generados mediante RS son expresiones en forma cerrada que comparten una estructura similar a la de los modelos de caja blanca, los cuales se derivan de principios fundamentales. El objetivo en RS es identificar la mejor forma de función y parámetros para un conjunto de datos dado utilizando operadores y funciones matemáticas comunes que sirven como bloques de construcción para la forma de la función (Koza, 1992). Esto contrasta con otras formas de análisis de regresión donde la forma de la función se pre-especifica y solo se optimizan los coeficientes numéricos a través del ajuste del modelo.

Sin embargo, es importante señalar que la RS es fundamentalmente un enfoque basado en datos. No requiere información previa sobre el proceso o sistema modelado, lo que da lugar a modelos de caja negra que, en ocasiones, pueden ser difíciles de interpretar. Los algoritmos de RS favorecen expresiones más cortas y utilizan simplificaciones que permiten un análisis detallado de los modelos.

Por estas razones, el uso conjunto de modelos subrogados y la RS es indispensable. Estos modelos capturan el comportamiento de sistemas más complejos o costosos de manera simple, eficiente y con la mínima pérdida de información. Los modelos subrogados sirven como sustitutos de simulaciones numéricas o modelos más difíciles de interpretar, permitiendo explicar de manera más clara los mecanismos de decisión y reduciendo el alto costo computacional asociado

a modelos complejos.

El objetivo de encontrar una función interpretable es una necesidad recurrente en las ciencias naturales y físico-químicas, donde los altos niveles de ruido y la variabilidad entre sistemas hacen crucial la dependencia de fuentes externas de conocimiento previo. En estos contextos, es preferible encontrar un modelo *biológicamente plausible* o *físicamente coherente* en lugar de uno que simplemente maximice la precisión de las predicciones.

Un modelo es considerado interpretable si la relación entre sus entradas y salidas puede ser trazada de manera lógica o matemática de forma concisa. Es decir, los modelos son interpretables cuando se expresan como ecuaciones matemáticas claras. En disciplinas como las ciencias naturales y físico-químicas, los modelos derivados de principios fundamentales permiten razonar sobre los fenómenos subyacentes (Makke y Chawla, 2024).

Al generar un modelo subrogado resolviendo problemas de RS, no solo se obtiene una representación simplificada del comportamiento de un sistema o proceso, sino que también se facilita la incorporación de conocimiento previo. Esto permite una mejor alineación entre el modelo y las necesidades de comprensión y generalización en las ciencias naturales y físico-químicas. Además, en disciplinas críticas como la salud, los modelos no interpretables, por muy precisos que sean, pueden no ser aprobados para su implementación (Mozafari-Kermani, Sur-Kolay, Raghunathan, y Jha, 2015), lo que refuerza la importancia de la interpretabilidad en entornos donde la transparencia es fundamental.

La mayoría de los algoritmos más conocidos para la RS incluyen variantes de Programación Genética basada en árboles (PG) (Koza, 1992), PG lineal (Oltean y Grosan, 2003) que abarca también el *Pareto-GP* (Smits y Kotanchek, 2005), PG cartesiana (Miller y Harding, 2008), evolución gramatical (GE, según su sigla en inglés) (O'Neill y Ryan, 2001) y programación de expresión genética (GEP, según su sigla en inglés) (C. Ferreira, 2001). Los desarrollos más recientes en RS comprenden PG con semántica geométrica (Moraglio, Krawiec, y Johnson, 2012; Pawlak y Krawiec, 2018; Ruberto, Terragni, y Moore, 2020), PG de regresión múltiple (Arnaldo, Krawiec, y O'Reilly, 2014) y un algoritmo evolutivo memético que emplea una representación fraccionaria novedosa para la RS (Sun y Moscato, 2019). Además, existen varios algoritmos no evolutivos para la RS como la extracción rápida de funciones (FFX, según su sigla en inglés) (McConaghy, 2011), *SymTree* (de França, 2018) y enumeración gramatical priorizada (PGE, según su sigla en inglés) (Worm y Chiu, 2013). Recientemente, se han utilizado diversas arquitecturas de redes neuronales para la RS (Sahoo, Lampert, y Martius, 2018; Kim y cols., 2019; Petersen y cols., 2019; Udrescu y Tegmark, 2020).

2.2.1. Ejemplo de problema de RS

Para explicar un problema de RS, se puede pensar en un sistema S que produce salidas (y) según una fórmula desconocida:

$$y = e^{-(x-\mu)^2/\sqrt{2\pi\sigma^2}}$$

El objetivo es modelar y para comprender la relación entre la entrada x y la salida y. Un modelo de regresión se entrenaría con un conjunto de datos (x, y), donde x es la entrada, mientras que y es la respuesta correspondiente. Después del entrenamiento, el modelo estará ajustado a los datos y podrá predecir la respuesta y_i para nuevas entradas x_i. La representación del modelo es:

$$\hat{y} = f(x; \theta)$$

Aquí, θ representa los parámetros del modelo entrenado y f es la función que lo representa. Sin embargo, este enfoque no permitiría entender la relación entre x y las variables internas del sistema, como μ y σ.

En contraste, si se usara un modelo de RS, se entrenaría con el mismo conjunto de datos (x, y), pero además de ajustarse a los datos, también proporcionaría una expresión escrita que relaciona x con y. Por ejemplo:

$$\hat{y} = e^{-(x-0{,}997)^2/2{,}5}$$

Aquí, asumiendo que los valores reales de μ y σ son ambos 1, el modelo proporcionaría una expresión donde $\mu = 0{,}997$ y $\sqrt{2\pi\sigma} = 2{,}5$, lo que muestra que los valores de y disminuyen exponencialmente con x y que esta relación depende de ciertas constantes. Toda esta información se perdería al usar solo un modelo de regresión estándar.

En este caso, el problema de ajuste de parámetros sería encontrar los valores óptimos para los coeficientes, mientras que el problema estructural implicaría determinar las funciones (por ejemplo, exponencial, división, potencia, resta) y su orden para construir la expresión adecuada.

2.3. Algoritmos Evolutivos

La idea de los algoritmos evolutivos (AEs) surgió en la década de los 60s gracias a John Holland y sus colegas de la Universidad de Michigan, como se describe en *An Introduction to Genetic Algorithms* (Mitchell, 1996). Su objetivo era estudiar formalmente el proceso de la evolución natural y desarrollar métodos para imitar los mecanismos de adaptación natural en sistemas informáticos.

Los AEs, inspirados en los principios de la evolución biológica, se han establecido como métodos eficaces para la optimización y la búsqueda de soluciones. Estos algoritmos mantienen un conjunto de entidades, conocidas como *individuos* o *cromosomas*, que representan soluciones potenciales del problema que se quiere resolver. A través de interacciones y competencias, estas entidades evolucionan, permitiendo que las mejores prevalezcan y conduzcan a soluciones cada vez más refinadas.

Este enfoque se enmarca en la computación evolutiva, una subrama de la inteligencia artificial, y es particularmente útil en escenarios donde el espacio de búsqueda es amplio y no lineal. En contextos donde los métodos convencionales fallan en encontrar soluciones en tiempos razonables, los AEs destacan por su robustez. Esto es posible debido al uso de operadores evolutivos, que son

estocásticos en lugar de deterministas, como se explica en Deb (2001). Los operadores estocásticos, encargados de combinar y alterar soluciones, evitan sesgar la búsqueda hacia una región específica del dominio, lo que permite a los AEs identificar múltiples óptimos cuando la función objetivo los presenta. De esta forma, proporcionan una perspectiva global durante la búsqueda, ayudando a resolver problemas complejos de manera más eficiente.

Adoptando términos de la teoría evolutiva, la población de individuos es sometida a operadores genéticos como el cruzamiento, que combina la información genética de dos o más individuos; la mutación, que introduce cambios aleatorios en los individuos para aumentar la diversidad genética; y la selección, que determina qué individuos prevalecerán en la siguiente generación. Operadores estocásticos como la selección por torneo (Deb, 2001) permiten que los mejores individuos sean seleccionados con mayor probabilidad.

Un algoritmo evolutivo puede presentar diversas variaciones, dependiendo de cómo se decide el reemplazo de los individuos para formar la nueva población, el o los criterios de parada que se elijan, entre otros. En general, el pseudocódigo consiste de los siguientes pasos:

Algorithm 1 Algoritmo Evolutivo Genérico

Input: Tamaño de la población N, Número de generaciones G, Función de evaluación f
Output: Mejor individuo encontrado

1: $P \leftarrow$ inicializarPoblación(N) ▷ Generar población inicial
2: **for** $g = 1$ to G **do** ▷ Iterar sobre generaciones
3: evaluarPoblación(P, f) ▷ Evaluar aptitud de cada individuo
4: $S \leftarrow$ seleccionar(P) ▷ Seleccionar mejores individuos
5: $H \leftarrow$ aplicarCruzamiento(S) ▷ Aplicar operador de cruzamiento
6: $P' \leftarrow$ aplicarMutación(H) ▷ Aplicar mutación a descendientes
7: $P \leftarrow$ reemplazar(P, P') ▷ Reemplazar con nueva población
8: **end for**
9: **return** mejorIndividuo(P) ▷ Devolver el mejor individuo

2.3.1. Programación Genética

La Programación Genética (PG), es una técnica de computación evolutiva que se inspira en los procesos de evolución biológica. En este caso, pueden ser tanto programas, expresiones matemáticas, u otro tipo de representación básica de la solución del problema a atacar. Esta metodología permite resolver una amplia gama de problemas de manera automática, y, a diferencia de otros enfoques de optimización, no requiere una definición completa de la estructura de la solución desde el principio (Koza, 1992, 1994).

Para poder representar un algoritmo de PG, es necesario definir una serie de elementos, y para poder ejecutarlo, se deben tomar algunas decisiones conocidas como *pasos preparatorios*, los cuales son presentados a continuación.

Primero, se debe definir la representación de los individuos de la población, que normalmente corresponden a expresiones matemáticas o funciones, se representan mediante árboles sintácticos. Se define el conjunto de terminales y el conjunto de funciones. El conjunto de terminales contiene aquellas entradas externas del algoritmo, por ejemplo, para el caso donde los individuos sean expresiones matemáticas, este puede ser un conjunto de constantes, variables y funciones sin argumentos. Por otro lado, el conjunto de funciones contiene las funciones que se utilizan en el algoritmo, pueden incluirse operaciones aritméticas, funciones matemáticas, booleanas, condicionales, bucles, entre otras, según las necesidades del problema a resolver.

Para poder evolucionar los individuos, se necesita partir de una población inicial. Esta población se puede definir de múltiples formas, siendo la más simple, generar la primera población de individuos de forma aleatoria.

El siguiente paso una vez que se tiene una población definida, es reproducir a la población para que evolucione de forma acorde a la solución deseada. Por tal motivo se define el operador de selección, el cual permite elegir, de entre todos los individuos de la población, aquellos que tienen una mayor aptitud para transmitir sus características a la siguiente generación. Este operador se basa en una función de aptitud (*fitness*), es decir, una métrica de evaluación que permita determinar qué tan *buenas* son las soluciones generadas por el algoritmo en términos de resolver el problema dado. Dicha función es utilizada por el operador de selección mediante criterios como la selección por torneo, selección por ruleta o selección elitista. De esta forma, se consigue una evolución gradual hacia soluciones más adaptadas para el problema a resolver.

Una vez que se seleccionan los individuos para la reproducción, se deben cruzar entre sí para generar descendencia, lo cual se realiza mediante operadores genéticos como el cruzamiento y la mutación. El cruzamiento consiste en intercambiar partes de los individuos seleccionados, mientras que la mutación introduce cambios aleatorios en los descendientes para explorar nuevas áreas del espacio de búsqueda. Para poder llevar a cabo la reproducción y mutación, previamente se deben establecer los parámetros de control del algoritmo, como el tamaño de la población, las probabilidades de realizar operaciones genéticas (cruzamiento, mutación), el tamaño máximo de los árboles, entre otros hiperparámetros. Además, se debe definir el criterio de parada, indicará cuándo detener la ejecución del algoritmo, lo que puede incluir un número máximo de generaciones, un criterio de éxito específico del problema o de convergencia. Además, el criterio suele especificar cual será el resultado final del algoritmo, que generalmente implica seleccionar el mejor individuo obtenido hasta ese momento.

A lo largo de las generaciones, la población y reproducción de programas, sumado a la naturaleza aleatoria y estocástica de la PG permite explorar de manera efectiva el espacio de búsqueda en busca de soluciones innovadoras y no convencionales que podrían no ser descubiertas por métodos deterministas, a la vez que se explota la información contenida en las buenas soluciones conseguidas en las generaciones anteriores.

Los métodos de PG utilizados para RS presentan una gran variedad de im-

plementaciones y estrategias a lo largo de los últimos años. Por ejemplo, *gplearn* sigue el modelo tradicional de PG con selección por torneo, mutación y cruzamiento mediante una API de scikit-learn sencilla de utilizar (Stephens, 2015). Los métodos de optimalidad de Pareto, como AFP (*Age-Fitness Pareto Optimization*), integran múltiples objetivos, incluyendo la edad para evitar la convergencia prematura y por supuesto el valor utilizado para medir el *fitness* de los individuos, en este caso, error promedio normalizado, para la mejora iterativa del modelo (M. Schmidt y Lipson, 2011; Hornby, 2006; M. Schmidt y Lipson, 2008). *EPLEX* y *SBP-GP* utilizan las semánticas de los programas para mejorar la selección y cruzamiento, respectivamente (Cava, Spector, y Danai, 2016; Virgolin, Alderliesten, y Bosman, 2019). Algoritmos como *FEAT*, *MRGP* y *FFX*, añaden estructuras específicas a los modelos (Cava, Singh, Taggart, Suri, y Moore, 2019; Arnaldo y cols., 2014; McConaghy, 2011), mientras que *GP-GOMEA* adapta los operadores de cruzamiento en el tiempo, de forma dinámica (Virgolin, Alderliesten, Witteveen, y Bosman, 2020).
Dichos algoritmos fueron comparados en el *benchmark* SRBench en su edición del año 2021 (La Cava y cols., 2021). Se presentarán más detalles de este *benchmark* en la Sección 4.4.1. Junto a estos algoritmos también se evaluó Operon (Burlacu, Kronberger, y Kommenda, 2019).

2.3.2. Operon

Junto a los anteriores algoritmos también se evaluó Operon (Burlacu, Kronberger, y Kommenda, 2019). Dicho algoritmo presentó los mejores valores de coeficiente de determinación (R^2), el cual mide qué tan bien las predicciones del modelo se ajustan a los datos observados. Asimismo, Operon destacó por generar expresiones de tamaño razonable y por requerir menos tiempo para encontrarlas, como se muestra en la Figura 2.2. En la competencia de 2022 del mismo *benchmark*, logró el cuarto puntaje más alto en la categoría de problemas *Real World*, enfocada a evaluar el desempeño de los modelos en problemas del mundo real, siendo el mejor modelo participante de tal competencia que únicamente utiliza PG (Burlacu, 2023).

Otro punto a favor, es que Operon posee un proyecto paralelo llamado *pyoperon*, el cual provee una interfaz de *scikit-learn* de uso sencillo implementada en Python que tiene la capacidad de realizar invocaciones a las operaciones implementadas en C++. Por estos motivos, se elige este *framework* como representante en lo que respecta a algoritmos puros de PG.

Operon es un *framework* de PG implementado en C++ con énfasis en el rendimiento, la modularidad y la facilidad de uso. Permite la expresión de diversas variantes de algoritmos genéticos basados en un generador de descendencia y un operador de transmisión. Posee una codificación de soluciones que fomenta el uso de caché, basada en el principio de localidad espacial mediante representación de los árboles de forma lineal, además, la evaluación está implementada teniendo en cuenta la localidad temporal mediante ejecuciones paralelas que utilizan operadores del mismo tipo.

A continuación, se resumen las principales características de Operon:

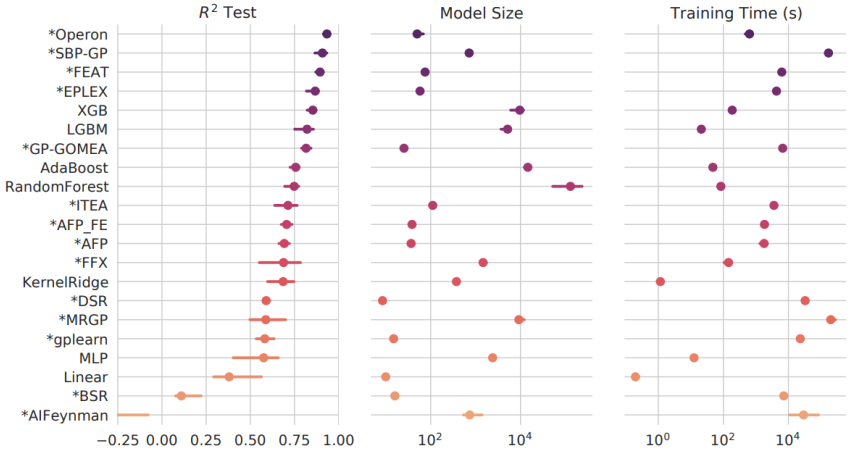

Figura 2.2: Comparativa sobre el *benchmark SRBench* del año 2021, columnas de izquierda a derecha: precisión, tamaño del modelo y tiempo de entrenamiento. Imagen extraída de La Cava y cols.(2021).

1. **Codificación de árboles:** Para representar un nodo, se define el tipo de datos escalar llamado *Plain old data type* o PODType, el cual, como se puede observar en la Figura 2.3, que cuenta con un diseño de memoria contigua que puede ser copiada, serializada o deserializada desde archivos binarios. Esto ofrece una velocidad de órdenes de magnitud mayor que la copia directa de objetos.

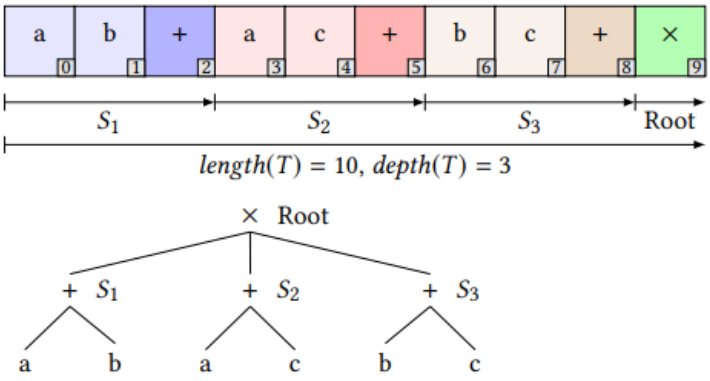

Figura 2.3: Codificación de los árboles en forma lineal mediante esquema de índices. Imagen extraída de Burlacu, Kronberger, y Kommenda (2019).

Para representar a un individuo (árbol), se utiliza un arreglo de nodos,

donde cada subárbol corresponde a un subarreglo delimitado por índices de inicio y fin. Las operaciones genéticas, como la inicialización, el cruzamiento y la mutación, aprovechan esta estructura lineal. Los nodos del árbol se organizan en un orden de postfijo (o recorrido postorden), de modo que durante la evaluación, los nodos hijos son procesados antes que sus nodos padres. Además, se definen propiedades como la longitud del subárbol y la profundidad del nodo para calcular eficientemente, en una sola pasada, todas las características necesarias de los nodos en el árbol.

2. **Inicialización de individuos:** Se define una clase para la inicialización que hace seguimiento de los nodos y sus probabilidades de elección a la hora de crear el nodo, permitiendo al usuario configurar la creación de los árboles. El algoritmo utilizado para crear árboles nuevos para la población es *Balanced Tree Creator* (BTC), que consiste en:

   a) Se define el conjunto primitivo $S = F \cup T$ en donde F es el conjunto de funciones y T el de terminales.

   b) Se obtiene un nodo aleatorio del conjunto primitivo, manteniendo un indicador de los nodos hijos vacíos. Observando la Figura 2.3, en una primera instancia se crea el operador ternario X (Root) con una aridad de 3, posteriormente, se completa recorriendo el árbol mediante búsqueda en anchura (BFS, por su sigla en inglés), en la que se recorren los niveles del árbol en orden como se puede observar en la Figura 2.4, manteniendo un registro de los nodos vacíos para conseguir una mayor eficiencia (por ejemplo, en la Figura 2.4, se almacenarán los nodos 8, 9, 10, 11 y 12 en dicho registro. Además, se puede notar que no se almacenan ni el 3 ni el 6 debido a que este algoritmo es una recorrida en orden BFS, por lo que se sabe que son hojas del árbol, normalmente variables o constantes). También se mantiene un indicador del número de nodos necesario para conseguir la altura objetivo del árbol.

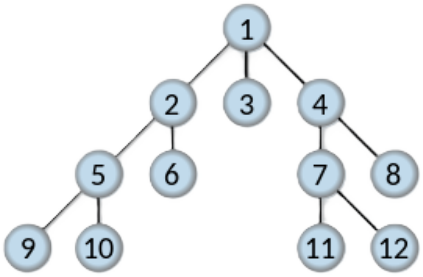

Figura 2.4: Orden BFS de exploración de un árbol. Imagen extraída de Drichel (2008).

c) Luego de obtener el nodo en el paso anterior, se selecciona un elemento del conjunto primitivo dependiendo de la probabilidad asignada, y se expande un nuevo nodo según la aridad de la función, por ejemplo, en la Figura 2.4, para el nodo 7 se eligió un operador binario y se crearon los nodos 11 y 12. Se limita según la diferencia entre la longitud objetivo y el numero de puntos de expansión por llenar. Cuando esta diferencia sea cero, se sustituirán los nodos por elementos terminales.

3. **Evaluación** La idea principal detrás de Operon es modelar la evaluación del problema mediante mínimos cuadrados no lineales. Para esto se define usando la biblioteca *Eigen* una matriz $m \times n$, con m igual al tamaño del *batch* (número de individuos a evaluar simultáneamente) y n el número máximo de nodos del árbol, y un vector de largo k, siendo k el número de puntos conocidos de la función *fitness*, es decir, los datos que se utilizan para evaluar el entrenamiento del algoritmo. Una vez se tiene el problema planteado de esta forma, se utiliza la biblioteca *Ceres* para resolverlo mediante mínimos cuadrados no lineales. Operon provee de distintas medidas de *fitness* para evaluar la calidad de las soluciones generadas, entre ellas el coeficiente de determinación (R^2). También se utilizan métricas como el *NMSE* (Error Cuadrático Medio Normalizado) y el *RMSE* (Raíz del Error Cuadrático Medio), que evalúan la magnitud del error en las predicciones. Más allá de la precisión, también se puede indicar a *Operon* que evalúe la complejidad del modelo mediante métricas como la longitud y la forma de la expresión generada.

   Finalmente, se permite la integración de funciones definidas por el usuario. Dicha evaluación está optimizada mediante el principio de localidad espacial, debido a la representación lineal del árbol (*PODType*) y localidad temporal, mediante la agrupación de operaciones del mismo tipo. Se implementa un paralelismo a nivel de datos mediante instrucciones simples con múltiples datos (SIMD, según su sigla en inglés), las cuales llevan a cabo la misma operación en múltiples cálculos de *fitness*. A su vez, se utiliza *batching* para explotar el paralelismo a nivel de datos.

4. **Reproducción**

   El algoritmo de Operon utiliza un componente generador de descendencia (*offspring*) que responde a las peticiones de otros componentes y devuelve un índice al individuo seleccionado de un lote (*batch*) de individuos. Existen tres métodos principales de selección:

   a) *Tournament selection*: Se eligen varios individuos al azar de la población y se realiza un torneo entre ellos, seleccionando al mejor como progenitor.

   b) *Proportional selection*: Cada individuo tiene una probabilidad de ser seleccionado basada en su *fitness*.

   c) *Rank-based selection*: Los individuos se ordenan según su *fitness* y se les asignan probabilidades de selección en función de su rango. A

diferencia de las otras dos, esta estrategia ayuda a evitar la dominación de individuos con *fitness* extremadamente alto, lo que permite ajustar mejor el equilibrio entre exploración y explotación.

En la programación genética clásica, la reproducción consiste en aplicar un proceso de selección seguido de cruzamiento (con posibles mutaciones), y la descendencia generada se acepta automáticamente en la población. Sin embargo, en Operon se introducen dos conceptos adicionales que mejoran este proceso:

a) *Brood generator*: Para cada pareja de padres, se generan varios hijos. A continuación, se realiza un torneo entre los descendientes y se selecciona al mejor para ser aceptado como el verdadero hijo.

b) *Offspring Selection Generator*: Los nuevos individuos solo se aceptan en la población si cumplen con ciertos criterios de aceptación, generalmente si el *fitness* del hijo supera al de ambos padres.

2.3.3. Optimización multiobjetivo

Cuando un problema de optimización involucra varias funciones objetivo, se denomina optimización multiobjetivo o toma de decisiones multicriterio. En este contexto, no es adecuado enfocarse únicamente en uno de los objetivos a expensas de los demás. Diversas soluciones pueden generar situaciones de compromiso o conflicto entre los distintos objetivos. Una solución que es óptima para un criterio puede no serlo para los otros, lo que hace inapropiado considerarla como la solución óptima global del problema, y surge la necesidad de alcanzar un equilibrio entre los diferentes objetivos (Deb, 2001).

2.3.3.1. Fundamentos de la optimización multiobjetivo

El modelo matemático general de un problema de optimización multiobjetivo (MO, según su sigla en inglés) se define en la siguiente ecuación:

$$\begin{aligned} &\text{mín} \,/\, \text{máx} : f_i(x), \quad i = 1, 2, \ldots, M \\ &\text{sujeto a:} \\ &g_j(x) \geq 0, \quad j = 1, 2, \ldots, J \\ &h_k(x) = 0, \quad k = 1, 2, \ldots, K \\ &x_l^{(l)} \leq x_l \leq x_u^{(l)}, \quad l = 1, 2, \ldots, n \end{aligned} \tag{2.1}$$

Para abordar este problema, es necesario establecer ciertos criterios que permitan identificar cuáles soluciones se consideran de buena calidad y cuáles no. En este contexto, se introduce el concepto de *dominancia*, que facilita la clasificación de las distintas soluciones y permite identificar alternativas óptimas teniendo en cuenta la presencia y la cuantificación de los M objetivos del problema.

Una solución $x^{(1)}$ domina a otra solución $x^{(2)}$ si se cumplen las siguientes condiciones:

1. La solución $x^{(1)}$ no es peor que $x^{(2)}$ en todos los objetivos.
2. La solución $x^{(1)}$ es estrictamente mejor que $x^{(2)}$ en al menos un objetivo.

Si alguna de las condiciones no se cumple, $x^{(1)}$ no domina a $x^{(2)}$.

El concepto de *dominancia* puede extenderse para identificar un conjunto de soluciones no dominadas dentro de una población. Si se considera una población de N soluciones, cada una con M valores de funciones objetivo, se puede emplear el siguiente procedimiento para encontrar el conjunto de soluciones no dominadas:

1. Fijar $i = 1$.
2. Para todos los $j \neq i$, comparar las soluciones $x^{(i)}$ y $x^{(j)}$ para determinar la dominancia, usando las dos condiciones mencionadas anteriormente para todos los M objetivos.
3. Si para algún j, $x^{(i)}$ es dominado por $x^{(j)}$, marcar $x^{(i)}$ como dominado. Incrementar i en uno e ir al paso 2.
4. Si todas las soluciones en el conjunto han sido consideradas, ir al paso 5; de lo contrario, incrementar i e ir al paso 2.
5. Todas las soluciones que no han sido marcadas como dominadas son soluciones no dominadas.

Cuando se realiza un análisis de dominancia entre dos soluciones y se encuentra que la primera condición de dominancia no se cumple para ninguna de las dos, no se puede concluir sobre la dominancia de una solución sobre otra. En este caso, se dice que las soluciones son *no dominadas*.

Si se tiene un conjunto finito de soluciones y se realiza una comparación exhaustiva de todos los pares posibles, se obtendrá un conjunto de soluciones que son no dominadas entre sí, y este conjunto tiene la propiedad de dominar al resto de las soluciones que no pertenecen a él. Por lo tanto, si se tiene un conjunto de soluciones P, el conjunto no dominado de soluciones P' está formado por aquellas que no son dominadas por ningún miembro del conjunto P, y a este conjunto se le denomina *frente de Pareto*. Cuando el conjunto P es todo el espacio de búsqueda, el conjunto no dominado P' resultante se denomina *frente óptimo de Pareto* (Flórez, Bolaños, y Cabrera, 2008).

En la Figura 2.5 se muestran los conjuntos óptimos de Pareto para diferentes escenarios con dos objetivos y para el mismo espacio de soluciones. En cualquier caso, el frente óptimo de Pareto siempre está compuesto por soluciones ubicadas en el borde de la región factible del espacio de soluciones.

Una población puede clasificarse en distintos niveles de no dominancia. Cuando el procedimiento de cinco pasos descrito previamente para encontrar

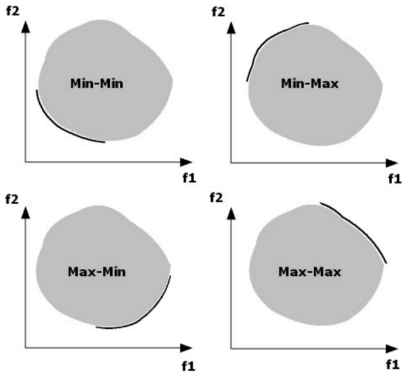

Figura 2.5: Frentes Óptimos de Pareto para el mismo espacio de soluciones. Imagen extraída de Flórez y cols. (2008).

el conjunto no dominado se aplica por primera vez a una población, el conjunto resultante corresponde al conjunto no dominado de mejor nivel, conocido como el *primer frente de Pareto*. Para obtener niveles adicionales de clasificación, las soluciones no dominadas pueden eliminarse temporalmente del conjunto original, permitiendo que el procedimiento sea aplicado nuevamente. Esto genera un nuevo conjunto no dominado de segundo nivel (o siguiente mejor nivel). Este proceso puede repetirse hasta que todos los miembros de la población sean clasificados dentro de algún nivel o frente de Pareto (Coello Coello, Lamont, y Van Veldhuizen, 2007).

2.3.4. Algoritmos Evolutivos multiobjetivo

Los algoritmos evolutivos multiobjetivo (MOEAs, según su sigla en inglés) surgieron en 1984 gracias a la propuesta de Schaffer, quien vio el potencial de utilizar AEs para abordar problemas con múltiples objetivos (Deb, 2001; Coello Coello y cols., 2007). La esencia de los MOEAs radica en su capacidad para trabajar simultáneamente con un conjunto de soluciones, lo que facilita su aplicación a problemas con múltiples criterios. En cada ejecución, el algoritmo aspira a hallar un conjunto de soluciones que se aproximen al frente de Pareto. Esto representa una importante ventaja respecto a los algoritmos tradicionales, que solamente generan una solución por ejecución. Complementariamente, los MOEAs tienen otras ventajas respecto a los algoritmos tradicionales, como ser menos sensibles a la forma o a la continuidad del frente de Pareto o permitir abordar problemas con espacio de soluciones de gran dimensión.

El objetivo principal del MOEA es acercarse al frente de Pareto y descubrir soluciones que representen diferentes equilibrios entre las funciones a optimizar. Esto facilita la toma de decisiones luego de la optimización. A diferencia de los AEs tradicionales, los MOEAs incorporan operadores adicionales, destacando el operador de diversidad. El primero busca evitar la convergencia prematura

hacia una región específica del frente de Pareto.

El objetivo principal de un MOEA es aproximarse al frente de Pareto y encontrar soluciones que representen diferentes equilibrios entre las funciones a optimizar, facilitando así la toma de decisiones al finalizar la optimización.

Los MOEAs se pueden clasificar en dos enfoques principales: los orientados a Pareto y los no orientados a Pareto. En los primeros, los algoritmos están diseñados desde el inicio para resolver problemas multiobjetivo, incorporando mecanismos de *ranking* y selección que mantienen la diversidad y aseguran una cobertura adecuada del frente de Pareto, buscando evitar la convergencia prematura hacia una región específica del mismo. Esta categoría incluye técnicas de segunda generación, que introducen elitismo mediante el uso de una población externa o el operador de selección. Los MOEAs no orientados a Pareto suelen tener un diseño más simple, en el cual la función de *fitness* se adapta para tener en cuenta múltiples objetivos. Entre los enfoques comunes para esta estrategia se encuentran la combinación lineal de los objetivos, asignando pesos *a priori* a cada uno, y la consideración de un objetivo como el principal, mientras que los restantes se tratan como restricciones (Goldberg y Holland, 1988).

Sin embargo, la complejidad inherente a los problemas de optimización multiobjetivo plantea un reto considerable, especialmente cuando aumenta la dimensión del espacio de soluciones. Los algoritmos exactos deterministas se vuelven ineficientes a medida que se incrementa la dificultad del problema. Estas técnicas clásicas, como los métodos *greedy* o *branch & bound*, suelen requerir un costo computacional muy grande para resolver problemas multiobjetivo de alta complejidad, como aquellos que se presentan en aplicaciones del mundo real.

Ante esta situación, las técnicas heurísticas estocásticas se han propuesto como una alternativa viable, proporcionando soluciones aproximadas de buena calidad en tiempos de cómputo razonables. Los MOEAs, en particular, emergen como una extensión de los AEs para problemas monoobjetivo, aprovechando conceptos relacionados con el tratamiento de funciones multimodales, lo que los convierte en métodos adecuados para abordar problemas con múltiples objetivos (Coello Coello y cols., 2007).

2.3.4.1. NSGA-II

Un método clásico es el algoritmo multiobjetivo NSGA-II, desarrollado por Kalyanmoy Deb en el año 2000 para superar varias limitaciones del algoritmo original NSGA (Deb, Pratap, Agarwal, y Meyarivan, 2002). Entre las críticas a NSGA se encontraban la falta de elitismo, la necesidad de ajustar el parámetro de *sharing*, y la alta complejidad computacional del ordenamiento no dominado, que era $O(MN^3)$, donde M es el número de objetivos y N es el tamaño de la población (Srinivas y Deb, 1994).

Para solucionar estas deficiencias, Deb *et al.* introdujeron el *fast non-dominated sorting*, que reduce la complejidad a $O(MN^2)$ al evitar comparaciones redundantes, aunque incrementa la complejidad espacial a $O(N^2)$ (Deb y cols., 2002). El NSGA-II también implementa elitismo, lo que permite preservar las mejores soluciones no dominadas en cada generación.

El proceso evolutivo de NSGA-II comienza con la combinación de la población de padres P_t y la población descendiente Q_t, ambas de tamaño N, para formar una población combinada R_t de tamaño $2N$. Esta nueva población se clasifica en frentes de Pareto mediante el ordenamiento no dominado rápido, comenzando con el frente más destacado (*primer frente de Pareto*, F_1), seguido por los frentes F_2, F_3, etc. De esta forma, F_1, contiene las soluciones de mejor rango, el segundo frente incluye individuos que son dominados solo por los individuos del primer frente, y así sucesivamente. Las soluciones se seleccionan de manera secuencial hasta llenar el espacio de la nueva población. Cuando el último frente excede el tamaño disponible, se utiliza la (*crowding distance*) para priorizar soluciones en regiones menos densas del espacio objetivo, favoreciendo la diversidad (Acampora, Kaymak, Loia, y Vitiello, 2013). La (*crowding distance*) $d_{I_j^m}$ para cada solución j, según un índice I, se calcula como una medida de dispersión entre vecinos:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\text{máx}} - f_m^{\text{mín}}}$$

donde $f_m^{\text{máx}}$ y $f_m^{\text{mín}}$ son los valores máximo y mínimo de la m-ésima función objetivo, y $f_m^{(I_{j+1}^m)}$ y $f_m^{(I_{j-1}^m)}$ son los vecinos de la solución j para cada una de las funciones objetivo m. A las soluciones extremas se les asigna una distancia infinita para preservar su inclusión, ya que están más alejadas en el espacio objetivo.

El elitismo se implementa mediante un esquema de selección tipo $(\mu + \lambda)$, en el que las poblaciones de padres y descendientes compiten por su lugar en la siguiente generación. Esto asegura que las mejores soluciones se mantengan a lo largo de las generaciones, lo que contribuye a la calidad de la población. Además, NSGA-II elimina la necesidad de parámetros de *sharing* mediante el *crowded-comparison operator*. Dicho operador clasifica soluciones no solo por su rango de dominancia, sino también por su *crowding distance*, favoreciendo así una mayor diversidad dentro de los frentes de Pareto, como ya fue mencionado (Flórez y cols., 2008).

Como se puede observar en la Figura 2.6, inicialmente, NSGA-II crea una población aleatoria P con un número predefinido de soluciones N. Luego, la evolución del algoritmo progresa a través de iteraciones sucesivas. En cada generación, se utilizan los operadores habituales de selección por torneo binario, cruzamiento y mutación para crear una población de descendencia Q de tamaño N. En el caso de problemas de RS, se suele utilizar operadores de cruzamiento y mutación de subárboles (Liu, Virgolin, Alderliesten, y Bosman, 2022).

2.3.5. jMetalPy

jMetalPy es un *framework* de *software* libre diseñado para el desarrollo, experimentación y estudio de algoritmos metaheurísticos para la optimización

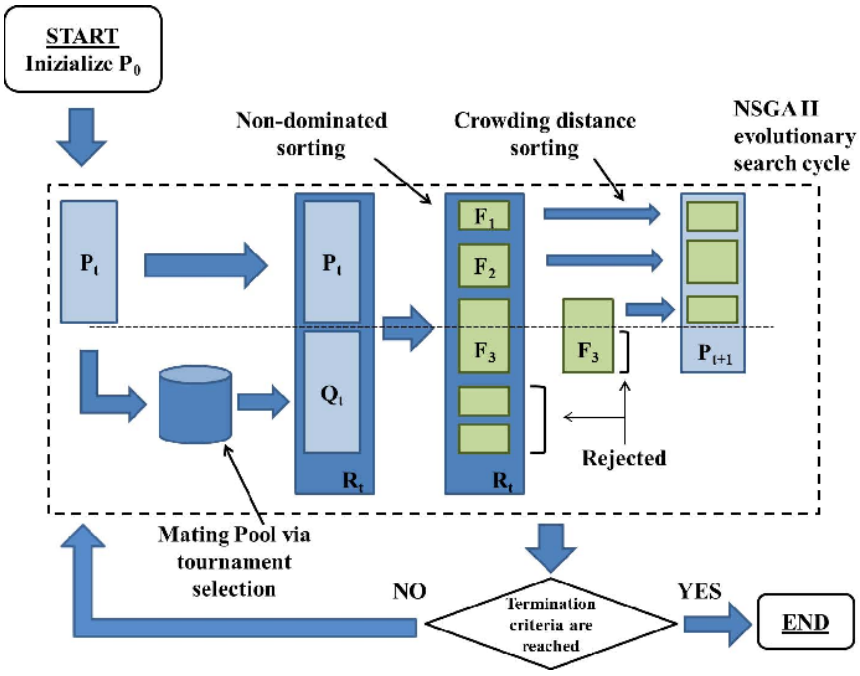

Figura 2.6: Estructura general de NSGA-II. Imagen extraída de Acampora y cols. (2013).

multiobjetivo (Benítez-Hidalgo, Nebro, García-Nieto, Oregi, y Del Ser, 2019). Implementado en Python, jMetalPy aprovecha su extenso ecosistema de bibliotecas para cálculo numérico y científico, como *NumPy*, *SciPy*, *Pandas*, y *Scikit-learn*. Además, ofrece herramientas de visualización como *Matplotlib*, *Holoviews* y *Plotly*. También brinda soporte para la computación paralela, permitiendo la evaluación de soluciones en paralelo a través de *Apache Spark* y *Dask*.

El objetivo de jMetalPy no era simplemente reescribir jMetal en Python según su propios creadores, sino también aprovechar las ventajas que este lenguaje ofrece en áreas no cubiertas por la versión original en Java. Entre ellas destacan la visualización interactiva en tiempo real, la articulación de preferencias para la toma de decisiones, y la resolución de problemas dinámicos.

jMetalPy incluye una gran colección de algoritmos de optimización multiobjetivo, como *NSGA-II*, *GDE3*, *SMPSO* y *MOEA/D*, además de versiones dinámicas de *NSGA-II* y *SMPSO* para problemas de optimización en entornos cambiantes. También soporta la resolución de problemas de referencia ampliamente utilizados, como *ZDT, DTLZ, WFG* y *LZ09*, los cuales permiten a los usuarios evaluar el rendimiento de los algoritmos o sus distintas configuraciones en problemas de *benchmarking*.

El *framework* ofrece una implementación de indicadores de calidad, como el *Hypervolume*, *Spread*, *Additive Epsilon*, *Inverted Generational Distance*, *Ge-*

nerational Distance, entre otras, las cuales resultan muy útiles para evaluar la diversidad y proximidad de las soluciones al frente de Pareto. Además, jMetalPy proporciona herramientas avanzadas de visualización para representar gráficamente los frentes de Pareto, tanto en problemas con dos objetivos (mediante gráficos de dispersión) como en aquellos con tres (gráficos 3D), o incluso en problemas con muchos objetivos (gráficos de coordenadas paralelas y diagramas de Chord personalizados).

jMetalPy también facilita la realización de estudios comparativos exhaustivos entre diferentes algoritmos. Incluye pruebas estadísticas como el Test de *Wilcoxon* y otras pruebas no paramétricas. Además, el *framework* ofrece la facilidad de generar automáticamente tablas en formato *LaTeX* con estadísticas clave como la media, desviación estándar, mediana y rango intercuartílico, lo que simplifica el análisis de los resultados (Benítez-Hidalgo y cols., 2019).

2.3.5.1. Arquitectura de jMetalPy

La arquitectura de jMetalPy está diseñada de forma orientada a objetos, lo que la hace flexible y extensible. La clase principal, `Algorithm`, gestiona una lista de soluciones y contiene un método `run()` que implementa el comportamiento de una metaheurística genérica. Este método incluye la creación inicial de soluciones, su evaluación y un bucle principal que ejecuta pasos hasta cumplir una condición de parada. Para comunicar el estado del algoritmo, se adopta el patrón de diseño *Observer*, permitiendo a los *observers* registrados recibir información relevante, como el número de evaluaciones o el tiempo de ejecución.

La clase `Problem` es responsable de crear y evaluar soluciones, caracterizándose por su número de variables de decisión, objetivos y restricciones.

Los operadores, como `Mutation`, `Crossover` y `Selection`, tienen un método `execute(source)`, que modifica o combina soluciones para generar nuevas. La clase `Solution` representa las codificaciones de las soluciones y contiene listas de variables y valores objetivos, junto con un conjunto de atributos.

En la Figura 2.7 se puede ver un diagrama de clases UML que resume la arquitectura anteriormente descrita.

2.4. Redes neuronales para regresión simbólica

2.4.1. Redes neuronales

Las redes neuronales profundas (DNN, según su sigla en inglés) son un tipo de modelo utilizado en el aprendizaje profundo, cuyo objetivo principal es aproximar una función f^*. Estos modelos se caracterizan por la dirección unidireccional del flujo de información, que va desde la entrada x hasta la salida intermedia que define f, y finalmente produce una salida deseada y. El término *redes* se refiere a que estas estructuras se representan comúnmente como una composición de múltiples funciones. Una forma típica de representar una red neuronal es a través de un grafo dirigido acíclico, que muestra cómo estas funciones están conectadas entre sí y cómo fluye la información a través de ellas.

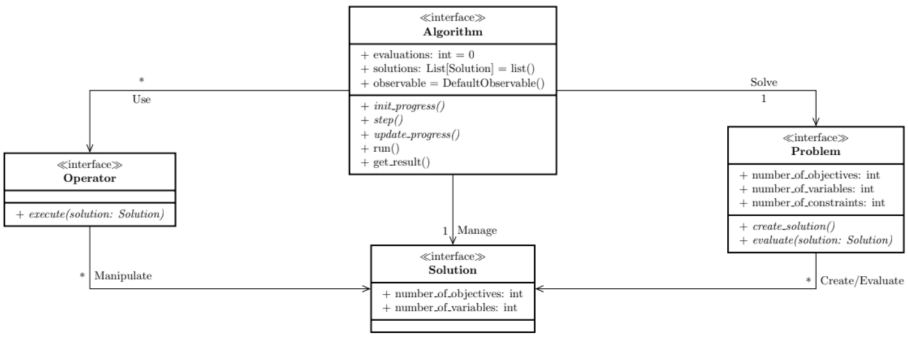

Figura 2.7: Diagrama de clases UML de jMetalPy. Imagen extraída de Benítez-Hidalgo y cols. (2019).

Durante el entrenamiento de las redes neuronales, el objetivo principal es ajustar el modelo $f(x)$ para que se asemeje lo más posible a la función deseada $f^*(x)$. Para lograr esto, se utilizan datos de entrenamiento que proporcionan una aproximación ruidosa de la función deseada en varios puntos del dominio de entrenamiento.

Para el entrenamiento se tiene un conjunto de datos (x, y), donde x es la entrada, mientras que y es la respuesta correspondiente. Esto permite que los ejemplos de entrenamiento indiquen cómo debe comportarse la capa de salida en cada punto x para producir un valor cercano a y. Así, el algoritmo de entrenamiento es responsable de descubrir cómo utilizar estas capas para obtener la salida deseada.

Las DNN introdujeron el concepto de capas ocultas, donde se emplean funciones de activación no lineales para calcular la salida de estas capas. Esto impide que las capas subsiguientes puedan reducirse fácilmente a una sola capa, ya que si no existieran estas funciones de activación o fueran funciones lineales, todas las capas podrían fusionarse en una sola capa.

Las redes neuronales son entrenadas mediante el método de retropropagación del gradiente, que emplea la regla de la cadena para calcular el gradiente de toda la red. Este enfoque es factible debido a que las conexiones en estas redes son secuenciales, lo que significa que solo se requiere la aplicación de la regla de la cadena para calcular los gradientes de la función de pérdida respecto a todos los parámetros.

El método de descenso del gradiente se utiliza para minimizar el valor de la función de pérdida. Sin embargo, no garantiza la convergencia a un óptimo global. En su lugar, la convergencia depende de la convexidad de la función de costo, así como de la inicialización de los parámetros de la red (Mejía, 2023).

2.4.2. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN, según su sigla en inglés) son una arquitectura de red neuronal diseñada para manejar datos secuenciales. A diferen-

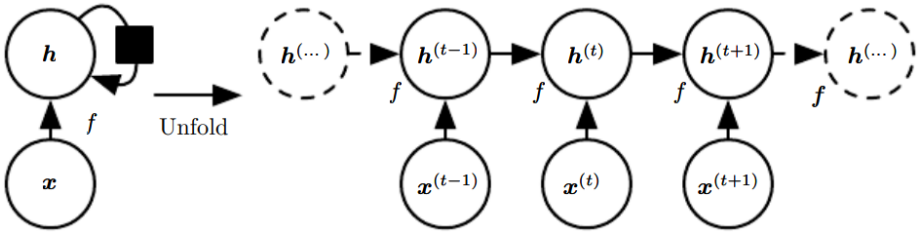

Figura 2.8: Una red recurrente sin salidas que procesa información de la entrada x y la incorpora en el estado h a lo largo del tiempo. (Izquierda) Diagrama del circuito con un retraso de un paso de tiempo. (Derecha) Gráfico computacional desplegado donde cada nodo representa un instante particular. Imagen extraída de Goodfellow y cols. (2016).

cia de las redes neuronales tradicionales *feedforward*, las RNN tienen conexiones que forman ciclos en la red, lo que les permite mantener un estado interno o memoria que puede ser utilizado para procesar secuencias de entradas.

Una de las principales características de las RNN es su capacidad para conservar información relevante de entradas anteriores a través de una memoria a corto plazo. Esta memoria es esencial para procesar datos en los que el orden de las entradas es significativo. La estructura de las RNN incluye bucles que permiten que el estado de la red en un momento dado influya en los estados futuros, lo que les confiere la habilidad para captar y recordar patrones en secuencias de datos (R. M. Schmidt, 2019).

En la Figura 2.8, se muestra una red recurrente sin salidas, que procesa información de la entrada x e incorpora esta información en el estado h a lo largo del tiempo. El diagrama del circuito (izquierda) ilustra un retraso de un paso de tiempo, mientras que el gráfico computacional desplegado (derecha) representa cada nodo como un instante particular.

Las RNN son ampliamente utilizadas en una variedad de aplicaciones que requieren el manejo de datos secuenciales. Entre estas aplicaciones se encuentran el procesamiento de lenguaje natural, donde se emplean para tareas como la traducción automática y el análisis de sentimientos; el reconocimiento de voz, en el cual se utilizan para transcribir y entender el habla; y el análisis de series temporales, donde se aplican para prever tendencias y patrones en datos temporales.

Una de las limitaciones de las RNN tradicionales es el problema del desvanecimiento o explosión del gradiente, lo que puede dificultar el aprendizaje en secuencias largas. Para abordar esta limitación, se han desarrollado variantes como las *Long Short-Term Memory* (LSTM) y las *Gated Recurrent Units* (GRU), que introducen mecanismos de compuerta para controlar el flujo de información y mejorar la capacidad de la red para capturar dependencias a largo plazo en los datos secuenciales (Mejía, 2023).

2.4.3. Modelo *Transformer* - Symformer

En 2017, un grupo de investigadores de Google publicó un trabajo (Vaswani y cols., 2023) en el que presentaron un modelo de aprendizaje automático diseñado para superar algunas limitaciones de las arquitecturas basadas en RNN y redes convolucionales (CNNs, según su sigla en inglés) en tareas de procesamiento de lenguaje natural (NLP, según su sigla en inglés), como la traducción automática. Este modelo demostró ser eficaz tanto en conjuntos de datos grandes como pequeños. Inspirado en varios enfoques previos de NLP, se basa en un mecanismo de codificación y decodificación (*encoder-decoder*), donde el modelo primero codifica la entrada y luego la decodifica para generar la salida.

Se destaca este modelo por ser el primero en utilizar exclusivamente el mecanismo de *Self-Attention* para la traducción de secuencias. Este mecanismo se basa en el relacionamiento de las diferentes posiciones de una secuencia de entrada para computar una representación de la misma.

Por lo tanto, este enfoque permite que a cada secuencia de entrada se le corresponda una representación numérica sobre un espacio de representaciones de una dimensión definida. Para llevar las secuencias a esta representación y traducirlas a otro idioma, lógicamente se utilizan dos arquitecturas, el codificador y el decodificador, cuya estructura se representa en la Figura 2.9.

2.4.3.1. Symformer

Basado en el modelo *Transformer* y en la evolución del estado del arte de la RS con una participación importante de las redes neuronales, el *framework SymFormer* intenta resolver el problema de la RS para funciones de 1 y 2 variables (Vastl, Kulhánek, Kubalík, Derner, y Babuška, 2024).

Además de su alta precisión, la velocidad de inferencia de este tipo de modelos es rápida, ya que no necesita el entrenamiento *in situ* del modelo sino que puede ser preentrenado una única vez. Este trabajo presenta una solución a una desventaja significativa de los *transformers* previos en la RS: la generación de constantes. Dado que, por la estructura *clave-valor* de un *transformer*, se generan solo valores conocidos, es decir, valores que solo entran dentro del vocabulario definido en el entrenamiento, esto se soluciona mediante un descenso del gradiente posterior a la generación de las ecuaciones. En este proceso, se permite a la expresión ajustar sus constantes, como se puede observar en el paso 3 de la Figura 2.11.

La arquitectura del modelo *SymFormer* se presenta en la Figura 2.10, la cual consta de dos funciones, una llamada *encoder* y la otra *decoder*. El *encoder* toma como entrada los puntos de datos, cada punto se pasa primero por una capa para aumentar la dimensionalidad al tamaño oculto del *transformer* y los vectores resultantes se concatenan para formar la secuencia que procesa el *encoder*. La secuencia de vectores ocultos pasa a través de cuatro bloques de *induced set attention*, cada uno compuesto por dos capas de *cross-attention* y dos bloques de redes *feed-forward*. La salida de cada capa se suma con su entrada antes de pasarla por una capa de normalización.

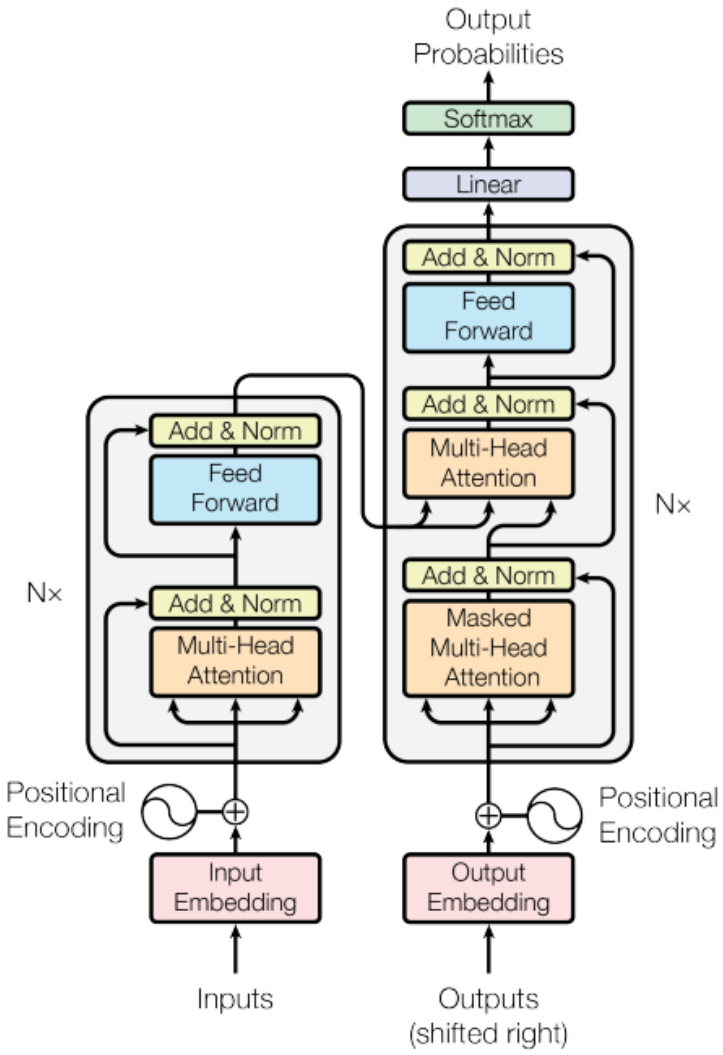

Figura 2.9: Arquitectura genérica de una red de tipo *Transformer*, a la izquierda está el *encoder* o codificador y a la derecha el *decoder* o decodificador. Imagen extraída de Vaswani y cols.(2017).

El *decoder* recibe una secuencia de *positional embeddings* entrenables que se suman con los símbolos de salida reales y se concatenan con las constantes proyectadas. La secuencia pasa por cuatro bloques estándar de *transformer*, constituidos por una capa de *self-attention*, una de *cross-attention* con la salida del *encoder* y un bloque *feed-forward*. Finalmente, se aplican dos cabezas: una de clasificación que produce los símbolos y otra de regresión que produce los valores de las constantes.

Como se puede ver en la Figura 2.11, en el paso 1 se representan los valores de entrada y de salida, estos son codificados y decodificados utilizando los pesos preentrenados en el paso 2 y posteriormente en el paso 3 se obtiene desde la salida del decodificador, una expresión la cual es sometida la búsqueda mencionada previamente.

2.4.3.2. Desventajas

Este método presenta dos desventajas principales. En primer lugar, requiere un entrenamiento previo que genera pesos, los cuales están disponibles únicamente para problemas con una o dos variables. Para problemas con más de dos variables, sería necesario realizar un nuevo entrenamiento, adaptado a la cantidad de variables necesaria. Este entrenamiento adicional para múltiples variables demanda tiempos y recursos significativos de GPU, ya que el modelo actual requirió más de 33 horas de entrenamiento utilizando 8 Tarjetas gráficas NVIDIA A100 (Vastl y cols., 2024).

La segunda desventaja es que no contempla la optimización multiobjetivo, ya que utiliza una función de pérdida basada en la entropía cruzada, combinada con un error cuadrático medio ponderado por un hiperparámetro:

$$\mathcal{L} = \mathcal{L}_{class} + \lambda \mathcal{L}_{MSE}$$

Esta función obliga al modelo a encontrar un compromiso entre precisión en clasificación simbólica ($\mathcal{L}_{class}$) y ajuste numérico ($\mathcal{L}_{MSE}$) en una única dimensión. Al operar mediante inferencias y no permitir un entrenamiento, tiene la gran desventaja de no ser configurable para múltiples objetivos.

2.4.4. *Deep Symbolic Regression*

Deep Symbolic Regression (DSR) es un enfoque de RS basado en aprendizaje por refuerzo (RL, según su sigla en inglés), que utiliza RNNs o LSTM, una estructura especial de RNNs, para generar expresiones matemáticas (Petersen y cols., 2019).

DSR aprovecha la característica de probabilidad condicional en RNNs para aumentar la parsimonia y acelerar la eficiencia de exploración. Este método se basa en la idea de que las expresiones matemáticas pueden ser representadas como árboles de expresión, donde los nodos internos son operadores matemáticos y los nodos terminales son variables de entrada o constantes.

Dicha RNN genera una distribución sobre expresiones matemáticas, las cuales luego se muestrean de esta distribución, se instancian y se evalúan según su

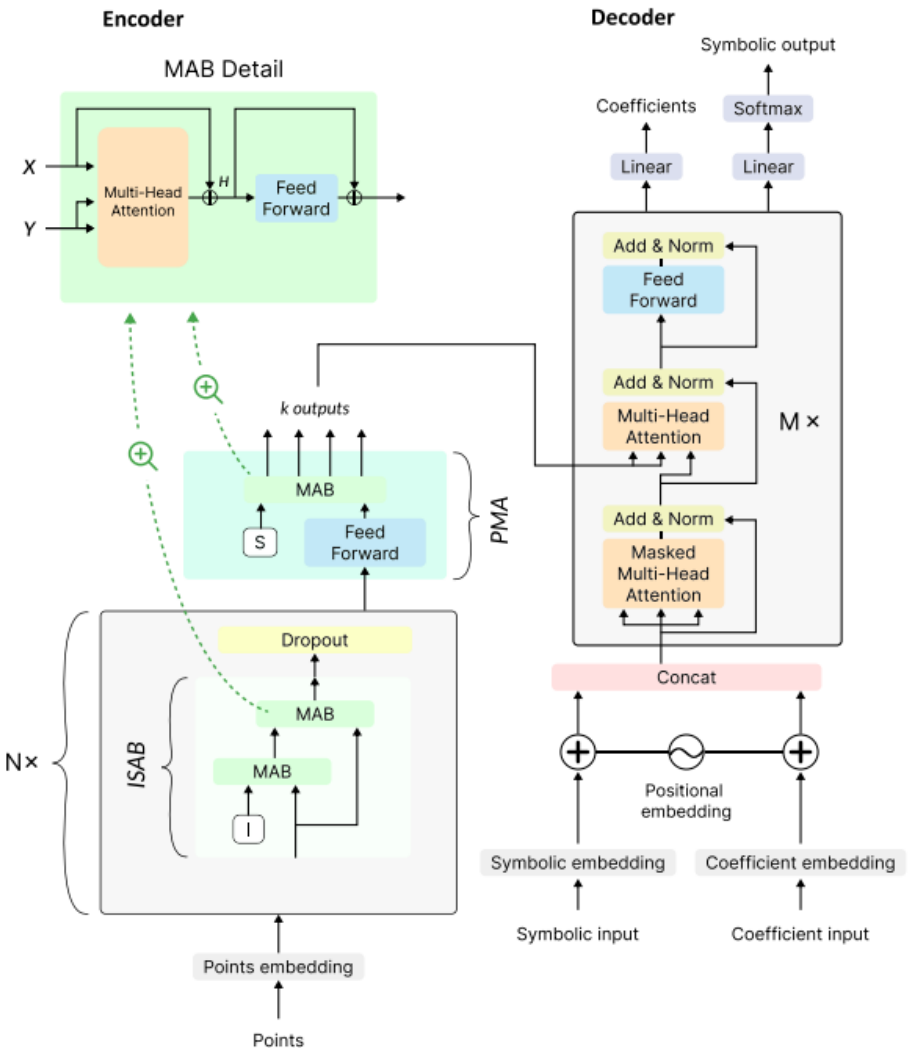

Figura 2.10: Esquema general de la arquitectura SymFormer. *MAB* se refiere al *Multihead Attention Block*, *ISAB* es el *Induced Set Attention Block* y *PMA* significa *Pooling by Multihead Attention*. Imagen extraída de Vastl y cols. (2024).

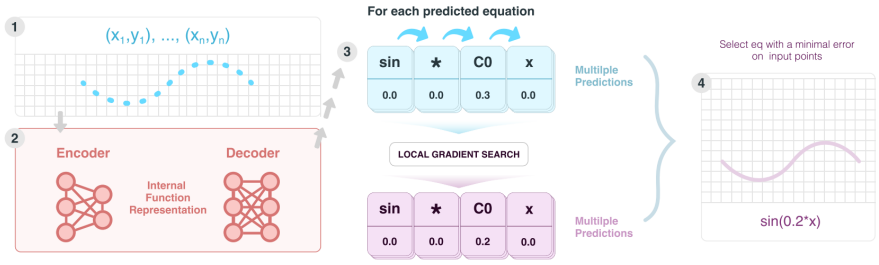

Figura 2.11: Diagrama de inferencia de Symformer. Imagen extraída de Vastl y cols. (2024).

ajuste al conjunto de datos. Esta adecuación se utiliza como señal de recompensa para entrenar la RNN mediante un algoritmo de gradiente de políticas orientado al riesgo (*risk-seeking policy gradient*). A medida que avanza el entrenamiento, la RNN ajusta la probabilidad de una expresión en relación con su recompensa, asignando mayores probabilidades a las mejores expresiones (Landajuela, Lee, Yang, y cols., 2022).

2.4.5. Generación de expresiones con una RNN

Como ya fue mencionado anteriormente, las expresiones matemáticas pueden representarse utilizando árboles de expresión simbólica. Estos árboles de expresiones se pueden representar como una secuencia de valores de nodos o *tokens*, mediante su recorrido en preorden. Esto permite generar un árbol de expresión secuencialmente manteniendo una correspondencia uno a uno entre el árbol y su recorrido.

Las expresiones generan un *token* τ_i a la vez a lo largo del recorrido en preorden (de τ_1 a τ_T). Una distribución categórica con parámetros ψ define las probabilidades de seleccionar cada *token* de una biblioteca L de *tokens* posibles. Para capturar el contexto de la expresión mientras se genera, esta probabilidad se condiciona en las selecciones de todos los *tokens* anteriores en ese recorrido. Esta dependencia condicional se puede lograr utilizando una RNN con parámetros θ que emite un vector de probabilidad ψ de manera autorregresiva.

El muestreo está restringido por restricciones fijas para limitar las combinaciones de *tokens* y las longitudes de las expresiones, es decir, las complejidades de las expresiones. Dado un rango de complejidad de expresión como restricción de exploración, la RNN detendrá el muestreo si la expresión tiene suficiente complejidad y el árbol de expresiones está completo. Este mecanismo busca evitar expresiones con un alto nivel de complejidad que pierdan interpretabilidad, que es el principal problema de la PG.

En la Figura 2.12 se ilustra un ejemplo del proceso de muestreo. Para cada *token*, la RNN emite una distribución categórica sobre los *tokens*, se muestrea un *token*, y el padre y el hermano del siguiente *token* se utilizan como la siguiente

entrada de la RNN. Los *tokens* siguientes se muestrean de forma autorregresiva hasta que todas las ramas del árbol alcanzan los nodos terminales. La secuencia de símbolos resultante es la del árbol, que puede utilizarse para reconstruir el árbol e instanciar su expresión correspondiente. Los colores corresponden al número de hijos de cada ficha. Los círculos blancos representan *tokens* vacíos. Los números indican el orden de muestreo de las fichas. B representa la biblioteca de *tokens*, mientras que C representa el árbol de expresiones muestreado en A. En este ejemplo, la expresión muestreada es: $sen(cx)/log(y)$, donde el valor de la constante c se optimiza con respecto a un conjunto de datos de entrada.

Dado que la salida de la RNN es simbólica, puede ser refinada aún más en base a la sintáxis de los símbolos. El procedimiento para imponer ciertos requisitos incluye restricciones sobre la longitud mínima y máxima de la expresión generada, evitando que todos los hijos de un operador sean constantes, y previniendo que el hijo de un operador sea su inverso (por ejemplo, evitando $\ln(\exp(x))$). Además, se excluyen operadores trigonométricos anidados en niveles profundos, ya que tales configuraciones ocurren raramente en aplicaciones científicas.

La optimización de constantes se realiza utilizando métodos estándar de la literatura de RS, buscando mejorar la precisión del modelo (Mundhenk y cols., 2021).

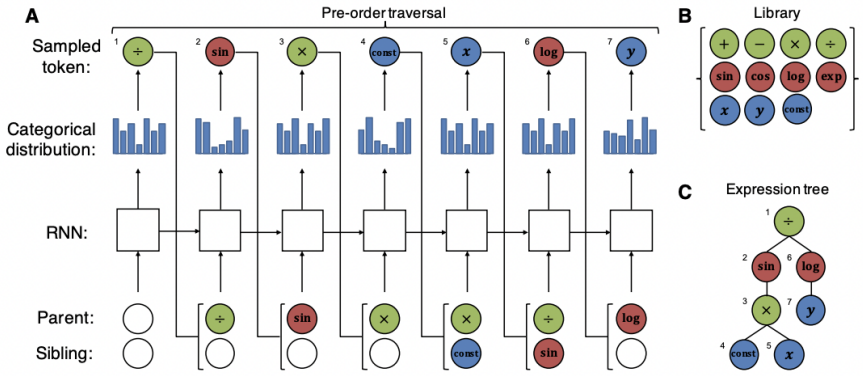

Figura 2.12: Ejemplo de muestreo de una expresión de la RNN. Imagen extraída de Petersen y cols. (2019).

2.4.6. Función de recompensa

En este trabajo, se hace una distinción entre la función de *recompensa* (o *reward*) y la función de *fitness*, dependiendo del contexto. La función de *fitness* se emplea comúnmente en problemas monoobjetivo y multi-objetivo, donde se evalúan soluciones en función de uno o más objetivos, respectivamente. En el caso monoobjetivo, la función de *fitness* mide la calidad de una única solución, mientras que en problemas multi-objetivo evalúa las soluciones en relación con

varios objetivos simultáneamente.

Por otro lado, la función de *recompensa* se utiliza en el contexto de RL. En este caso, la recompensa guía el proceso de optimización mediante retroalimentación, cuyo valor varía entre 0 y 1, ajustando las políticas del modelo conforme avanza el entrenamiento. La recompensa es el objetivo inmediato que el modelo busca maximizar o minimizar en RL.

En la Sección 2.5.2, se aborda cómo la RNN de DSR se entrena utilizando RL o un enfoque relacionado. Aquí, el término *recompensa* refleja lo que el algoritmo DSR minimiza directamente durante su proceso de aprendizaje. En cambio, el *fitness* se refiere a una métrica indirecta, como el error, que mejora a lo largo de las iteraciones, ya que es considerado en la función de recompensa. En resumen, mientras que la recompensa es el objetivo directo en RL, el *fitness* se optimiza indirectamente en procesos evolutivos o iterativos como medida de la calidad de la solución.

En DSR, una vez que se muestrea un recorrido en preorden, se instancia la expresión simbólica correspondiente y se evalúa con una función de *fitness*. Una medida de ajuste estándar en la RS basada en PG es el error cuadrático medio normalizado (NRMSE), que es el error cuadrático medio normalizado por la desviación estándar de los valores objetivo, σ_y. Es decir, dado un recorrido en preorden τ y un conjunto de datos de pares (X, y) de tamaño N, con $X \in \mathbb{R}^n$ y $y \in \mathbb{R}$, se define

$$\text{NRMSE}(\tau) = \frac{1}{\sigma_y}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(X_i))^2},$$

donde $f : \mathbb{R}^n \to \mathbb{R}$ es la expresión matemática instanciada representada por τ, y σ_y es la desviación estándar de y. Finalmente, la función de recompensa se define como $R(\tau) = \frac{1}{1+\text{NRMSE}(\tau)}$, la cual los autores también denominan como *Inverse NRMSE*. Esta varía entre 0 y 1.

2.4.7. Optimización de constantes

Si la biblioteca L incluye el *token* de constante, las expresiones muestreadas pueden incluir varios marcadores de posición de constante. Estos se pueden ver como parámetros ξ de la expresión simbólica, que se optimizan maximizando la función de recompensa: $\xi^* = \arg\max_\xi R(\tau; \xi)$, utilizando un algoritmo de optimización no lineal, por ejemplo, BFGS. Este bucle de optimización interna se realiza para cada expresión muestreada como parte del cálculo de recompensa antes de realizar cada paso de entrenamiento.

A continuación, en la sección 2.5.2 se consideran más aspectos de DSR, ya que es utilizado por otro método como un generador de secuencias, para brindarle a un algoritmo de PG, la población inicial.

2.5. Algoritmos híbridos - *Deep Symbolic Optimization*

El algoritmo *Deep Symbolic Optimization* (DSO) consta de dos componentes principales: un generador de secuencias (con parámetros aprendibles) y un componente de programación genética.

2.5.1. Preliminares

Como se introdujo en la sección referente a Programación Genética, cualquier expresión matemática f puede ser representada por un árbol de expresiones algebraicas, donde los nodos internos son operadores y los nodos terminales son variables de entrada o constantes (Mundhenk y cols., 2021). Al igual que en DSR, $\tau = [\tau_1, \ldots, \tau_{|\tau|}]$ se refiere al recorrido en preorden de dicho árbol de expresiones. Un aspecto a remarcar, es que existe una correspondencia uno a uno entre un árbol de expresiones y su recorrido en preorden (Landajuela y cols., 2021). Cada τ_i es un operador, una variable de entrada o una constante seleccionada de una biblioteca de posibles *tokens*, por ejemplo, $[+, -, \times, \div, \sin, \cos, \exp, \log]$. Un recorrido en preorden τ puede ser instanciado en una expresión matemática correspondiente f y evaluado en función de su ajuste a un conjunto de datos.

Nuevamente, los mismos autores de DSR consideraron la función de recompensa $R(\tau) = \frac{1}{1+\text{NRMSE}(\tau)}$.

2.5.2. Generador de secuencias

El generador de secuencias es una distribución parametrizada sobre expresiones matemáticas, $p(\tau|\theta)$. Típicamente, se elige un modelo de manera que la probabilidad de una expresión sea tratable con respecto a los parámetros θ, permitiendo la retropropagación de una función de pérdida diferenciable. Para ellos, se eligió un modelo RNN autorregresivo, en el cual la probabilidad del i-ésimo *token* (denotado τ_i) es condicionalmente independiente dado los *tokens* precedentes $\tau_1, \ldots, \tau_{i-1}$. Es decir, $p(\tau_i|\tau_{j\neq i}, \theta) = p(\tau_i|\tau_{j<i}, \theta)$. Siguiendo el generador de secuencias utilizado en DSR, esta RNN comprende una LSTM de una sola capa con 32 nodos ocultos.

Este generador de secuencias se entrena utilizando RL o un enfoque relacionado. Bajo esta perspectiva, el generador de secuencias puede ser visto como una política de RL, que se busca optimizar mediante el muestreo de un lote de N expresiones T, evaluando cada expresión bajo una función de recompensa $R(\tau)$, y realizando descenso del gradiente en una función de pérdida. En el trabajo referenciado, los autores exploraron tres métodos para entrenar el RNN:

- **Gradiente de política estándar (VPG)**: Utilizando la conocida regla *REINFORCE* (Williams, 1992), el entrenamiento se realiza sobre el lote T, dando como resultado la función de pérdida:

$$L(\theta) = \frac{1}{|T|} \sum_{\tau \in T} (R(\tau) - b) \nabla_\theta \log p(\tau|\theta), \tag{2.2}$$

donde b es un término base o variable de control, por ejemplo, una media móvil ponderada exponencialmente (EWMA) de recompensas.

- **Gradiente de política con búsqueda de riesgo (RSPG)**: (Landajuela y cols., 2021) desarrollan una alternativa a VPG destinada a optimizar para el mejor caso en lugar de la recompensa promedio:

$$L(\theta) = \frac{1}{\epsilon|T|} \sum_{\tau \in T} (R(\tau) - \tilde{R}_\epsilon) \nabla_\theta \log p(\tau|\theta) 1_{R(\tau) > \tilde{R}_\epsilon}, \tag{2.3}$$

donde ϵ es un hiperparámetro que controla el grado de búsqueda de riesgo y $\tilde{R}_\epsilon$ es el cuantil empírico $(1 - \epsilon)$ de las recompensas de T.

- **Entrenamiento de cola de prioridad (PQT)**: (Abolafia, Norouzi, Shen, Zhao, y Le, 2018) introducen un enfoque no basado en RL también destinado a enfocarse en optimizar el mejor desempeño. Las muestras de cada lote se agregan a una cola de prioridad de máxima recompensa persistente (MRPQ, según su sigla en inglés), y el entrenamiento se realiza sobre muestras en la MRPQ utilizando un objetivo de aprendizaje supervisado:

$$L(\theta) = \frac{1}{k} \sum_{\tau \in MRPQ} \nabla_\theta \log p(\tau|\theta), \tag{2.4}$$

donde k es el tamaño de la MRPQ.

2.5.3. Componente de Programación Genética

Para el componente de PG, los autores partieron de una formulación estándar del *framework DEAP* (Fortin, De Rainville, Gardner, Parizeau, y Gagné, 2012). Luego, introdujeron varios cambios clave:

1. En lugar de utilizar exclusivamente la mutación uniforme, se elige entre diferentes tipos de mutaciones: mutación uniforme, reemplazo de nodos, inserción o mutación por reducción, todas con la misma probabilidad.

2. Se incorporan restricciones propuestas por (Landajuela y cols., 2021). Por ejemplo, se restringen funciones trigonométricas anidadas, como: $\sin(1 + \cos(x))$. Si una operación genética genera un individuo que viola alguna restricción, dicha operación se revierte, convirtiendo la expresión del hijo en una copia de la del padre. Este enfoque de los autores busca garantizar que todos los individuos respeten las restricciones en cada generación.

3. La población de muestras nunca se inicializa aleatoriamente. En su lugar, la población inicial siempre se llena con muestras de la RNN.

2.5.4. Generación de población de Programación Genética guiada por redes neuronales

El enfoque híbrido que combina la búsqueda guiada por redes neuronales y la PG tiene como objetivo aprovechar las fortalezas de ambos métodos. Mientras que la PG es un proceso sin estado y no tiene un paso de aprendizaje, la búsqueda guiada por redes neuronales es un proceso con estado (a través de los parámetros θ de la RNN) y aprende de los datos mediante una función de pérdida bien definida y diferenciable. Sin embargo, se sabe que la búsqueda guiada por redes neuronales puede quedar atrapada en óptimos locales. En este contexto, el componente de PG puede introducir mayor diversidad en la población, lo que genera soluciones que se encuentran fuera de la distribución inducida por el RNN, ayudando a escapar de esos óptimos locales.

DSO combina la búsqueda guiada por redes neuronales y la PG, aprovechando las fortalezas de ambos métodos para mejorar la exploración del espacio de soluciones. La Figura 2.13 muestra un esquema general del proceso: un generador de secuencias, como una RNN, genera una población inicial de expresiones simbólicas que luego es utilizada por el componente de PG para realizar varias generaciones de evolución. Posteriormente, las mejores soluciones se seleccionan y utilizan para actualizar el modelo basado en la RNN, repitiendo este ciclo hasta alcanzar el número máximo de evaluaciones u otra condición de parada.

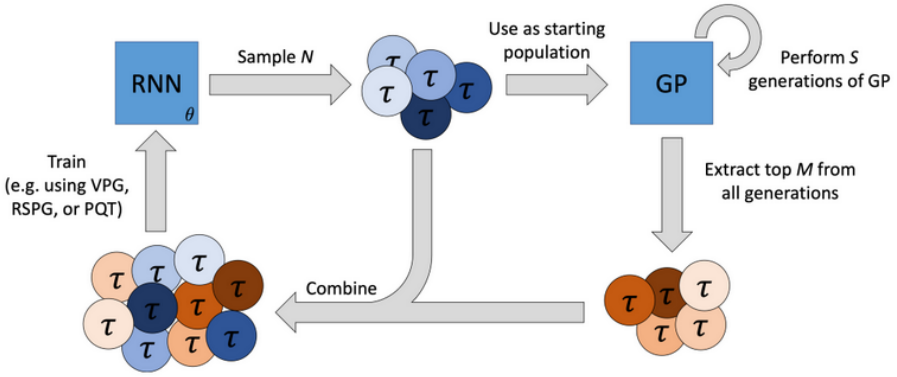

Figura 2.13: Esquema general del enfoque híbrido de DSO, que combina la generación de secuencias parametrizadas (por ejemplo, mediante una RNN) con PG para la resolución de problemas de RS. Imagen extraída de Mundhenk y cols. (2021).

DSO realiza esta interconexión a través de la población inicial de PG, $T_{GP}^{(0)}$. Específicamente, se utiliza el lote más reciente de muestras generadas por la RNN como la población inicial para PG: $T_{GP}^{(0)} = T_{RNN}$. A partir de esta población inicial, se llevan a cabo S generaciones de PG, lo que da como resultado una población final de PG, $T_{GP}^{(S)}$. Posteriormente, se sub-selecciona un *conjunto*

élite de las muestras de mejor desempeño de PG para ser utilizadas en la actualización de gradiente de la búsqueda guiada por redes neuronales (por ejemplo, VPG, RSPG, PQT). Este proceso es fundamental en el algoritmo y se repite hasta que se alcanza un número máximo de evaluaciones totales de expresiones u otra condición de parada.

Es importante señalar que el proceso de PG se reinicia para cada nuevo lote de muestras generadas por la RNN. Esto hace que el proceso sea similar a la PG con reinicios aleatorios. Sin embargo, la diferencia clave radica en que la población inicial de PG en cada reinicio cambia progresivamente a medida que el RNN aprende mediante una función objetivo. Desde la perspectiva de PG, el RNN proporciona poblaciones iniciales cada vez mejores, lo que facilita la convergencia del algoritmo. Empíricamente, se ha encontrado que este enfoque híbrido permite utilizar tasas de aprendizaje más grandes en comparación con la búsqueda puramente guiada por redes neuronales (Mundhenk y cols., 2021).

El algoritmo *Deep Symbolic Optimization* es considerado un método de vanguardia, tras haber alcanzado el primer lugar en la competencia SRBench 2022 en la categoría de aplicaciones del mundo real[1]. En esta competencia, los métodos participantes fueron entrenados para construir modelos predictivos interpretables para pronosticar los conteos de casos, hospitalizaciones y muertes por COVID-19 en un período de 14 días en el estado de Nueva York. Estos modelos fueron revisados por un experto en la materia, quien les asignó calificaciones de confianza (*trust ratings*), además de ser evaluados en términos de precisión y simplicidad.

DSO se ha aplicado en una amplia gama de aplicaciones, pero se ha visto obstaculizado por la la incapacidad de manejar múltiples objetivos de forma nativa (Faris y cols., 2024).

[1] https://cavalab.org/srbench/competition-2022/

Capítulo 3

Revisión de antecedentes

3.1. Introducción

En la última década, se han desarrollado numerosos trabajos que abordan la generación de modelos subrogados utilizando técnicas como la programación genética, redes neuronales, o enfoques híbridos que combinan ambas metodologías. Para la selección de los trabajos que se revisarán a continuación, se establecieron criterios clave basados en la relevancia para este proyecto, tanto en la resolución de problemas reales como en el uso de problemas de *benchmarking* significativos, así como en las técnicas de resolución empleadas. Estos trabajos han sido seleccionados por su contribución al campo, representando antecedentes exitosos y confiables. Además, han facilitado una comprensión más profunda de los conceptos implicados y las decisiones de diseño adoptadas en las diferentes fases del proceso. Aunque se ha consultado una gran cantidad de bibliografía, no toda esta literatura se incluye en la presente sección, dado que no cumple con los criterios específicos establecidos.

3.2. Trabajos relacionados

3.2.1. Deep learning para regresión simbólica

Varios enfoques recientes aprovechan el aprendizaje profundo para la RS. AI Feynman (Udrescu y Tegmark, 2020) propone una herramienta de simplificación de problemas para la RS. Utilizan redes neuronales para identificar propiedades de simplificación en un conjunto de datos (por ejemplo, separabilidad multiplicativa, simetría traslacional, entre otras), que explotan para definir recursivamente sub-problemas simplificados que luego se pueden abordar utilizando cualquier algoritmo de RS. AI Feynman utiliza técnicas inspiradas en la física para descomponer problemas complejos en partes más manejables, aplicando posteriormente métodos de ajuste basados en redes neuronales para descubrir relaciones matemáticas subyacentes. En el conjunto principal de 100 ecuacio-

nes del *dataset* Feynman, AI Feynman logra descubrir todas las ecuaciones, superando otros *software* como *Eureqa*, que solo lograba resolver 71 (Praksova, 2011). Estas ecuaciones se utilizaron como conjunto de entrenamiento para AI Feynman, permitiendo optimizar tanto la implementación del algoritmo como sus hiperparámetros, con el objetivo de maximizar su rendimiento. Para evaluar la generalización de AI Feynman, se utilizó un conjunto adicional de 20 ecuaciones conocidas como los *bonus mysteries*, los cuales eran considerablemente más complejos. Según los resultados en esta prueba de validación, *Eureqa* resolvió solo el 15 % de estos problemas, mientras que AI Feynman logró resolver el 90 %.

En GrammarVAE, Matt J Kusner y Hernández-Lobato (2017) desarrollan un modelo generativo para objetos discretos que se adhieren a una gramática preespecificada y luego los optimizan en el espacio latente. Demuestran que esto puede usarse para la RS, sin embargo, el método tiene dificultades para recuperar las expresiones exactas, y las expresiones generadas no siempre son sintácticamente válidas.

En Sahoo y cols. (2018), los autores desarrollan un marco de RS utilizando redes neuronales cuyas funciones de activación son operadores simbólicos, como la multiplicación o la división. Aunque este enfoque permite un sistema diferenciable de extremo a extremo, la retropropagación a través de funciones de activación simbólicas requiere simplificaciones al espacio de búsqueda, lo que limita la capacidad para aprender ciertas expresiones como $\sqrt{x}$ o $\sin(x/y)$. La anterior arquitectura es parte de un esfuerzo mayor para diseñar redes neuronales que contengan sesgos inductivos que las hagan más adecuadas para la exploración científica (Martius y Lampert, 2016).

También se han propuesto arquitecturas de aprendizaje profundo como PDE-Net y PDE-Net 2.0 (Long, Lu, Ma, y Dong, 2018; Long, Lu, y Dong, 2018) para predecir la dinámica de sistemas espacio-temporales y producir operadores diferenciales interpretables. Trask y cols. (2018) introdujeron la *Neural Arithmetic Logic Unit* (NALU), que introduce sesgos inductivos hacia operaciones aritméticas para mejorar la extrapolación en tareas específicas. Asimismo, redes neuronales han sido usadas para extraer parámetros interpretables de sistemas dinámicos (Zheng, Luo, Wu, y Tenenbaum, 2018; Lu, Kim, y Soljačić, 2019).

3.2.2. Métodos basados en *Transformers* generativos preentrenados (GPT)

Un *transformer* generativo preentrenado (GPT, por su sigla en inglés), es un modelo de lenguaje cuya arquitectura está basada en el modelo *Transformer* (Vaswani y cols., 2017), mencionado en la Sección 2.4.3 de este documento. Este tipo de modelo tiene la capacidad de generar texto coherente con una buena representación del contexto y realizar tareas clásicas de procesamiento de lenguaje natural, como traducción automática, resúmenes de texto, clasificación y más. Esto se puede llevar a cabo, incluso, mediante ajustes del *prompt* que se le presenta al modelo en la inferencia.

El proceso de inferencia del modelo SymbolicGPT (Valipour, You, Panju, y Ghodsi, 2021) se divide en tres pasos principales. Primero, se genera un vector

de *embedding* a partir de los datos de entrenamiento utilizando una red T-net, lo cual proporciona al modelo una representación de los datos de entrada invariante al orden. Esto implica que el modelo no obtiene información adicional del orden de los puntos en el conjunto de entrenamiento (Qi, Su, Mo, y Guibas, 2017).

A continuación, SymbolicGPT genera una "ecuación esqueleto", esto es, una ecuación cuyas constantes son reemplazadas por *placeholders* los cuales son modificados mediante una iteración para optimizar en función de la evaluación de dicha expresión con los datos de entrenamiento, retornando este nuevo resultado como salida.

Finalmente, cabe destacar que el algoritmo Symformer, presentado en capítulos anteriores, logró obtener en promedio mejores resultados en términos de R^2 comparado con DSO para 6 problemas de *benchmarking*, reduciendo además el tiempo de cómputo aproximadamente a una sexta parte de lo requerido por DSO (Vastl y cols., 2024).

3.2.3. Híbridos de Programación Genética y algoritmos de políticas de gradiente

Diversos autores han señalado que la idea de combinar PG con métodos basados en gradientes es anterior a la era del aprendizaje profundo (Igel y Kreutz, 1999; Topchy y Punch, 2001; Zhang y Smart, 2004; Montastruc, Azzaro-Pantel, Pibouleau, y Domenech, 2004; Wierstra, Schaul, Peters, y Schmidhuber, 2008).

Recientemente, varios enfoques han combinado PG y RL. En estos trabajos, el RL se utiliza para alterar o aumentar el proceso de PG, por ejemplo, ajustando las probabilidades de realizar diferentes operaciones genéticas. Alternativamente, la PG se utiliza para aumentar la creación o el funcionamiento de una red neuronal (Such y cols., 2018; Simyung Chang y Kwak, 2018; Gangwani y Peng, 2018; Yukang Chen y Wang, 2019; Stanley, Clune, Lehman, y Miikkulainen, 2019; Real, Liang, So, y Le, 2019; Miikkulainen y cols., 2019; Sehgal, La, Louis, y Nguyen, 2019; Tian y cols., 2020; Diqi Chen y Gao, 2020; Qiong Chen y Wang, 2020).

DSO, al que ya se dedicó una sección en este informe, destaca por su flexibilidad y eficiencia al aplicarse a la RS (Landajuela y cols., 2022). Por ejemplo, DSO ha sido integrado exitosamente con métodos de RS para mejorar la identificación paramétrica en modelos de vehículos submarinos (Wu, Wang, Ge, Wu, y Yang, 2017), así como en el desarrollo de modelos de turbulencia a través de técnicas que combinan programación genética con redes neuronales (H. Li, Waschkowski, Zhao, y Sandberg, 2023).

Además, el enfoque de DSO ha sido ampliado para incluir el uso de modelos de lenguaje en la RS, lo que permite acelerar el proceso de descubrimiento de soluciones (Da Silva, Goncalves, y cols., 2023). La capacidad de DSO para integrarse con otros métodos y formar enfoques híbridos se ha demostrado en diversas aplicaciones, como la fusión de información en el ámbito de la salud (Schnur y Chawla, 2023) y el diseño asistido por inteligencia artificial en convertidores de potencia (Da Silva y cols., 2023). Estas integraciones no solo amplían

el alcance de DSO, sino que también mejoran su rendimiento y aplicabilidad en problemas complejos, tales como los problemas de ingeniería de procesos.

3.2.4. Métodos de intercambio de poblaciones de muestras

Se pueden identificar una amplia clase de métodos que pueden caracterizarse mediante el uso de muestras de un generador de secuencias e intercambio de esas muestras con una población de PG. El generador de secuencias puede ser cualquier distribución discreta o proceso generativo que crea una secuencia de *tokens*, por ejemplo, una RNN o un *transformer*. Las muestras del generador de secuencias se tratan como intercambiables con una población generada por PG. Así, las muestras del generador de secuencias pueden ser insertadas en la población de PG y las muestras de PG pueden usarse para actualizar el generador de secuencias. En algunos casos, el generador de secuencias puede no tener parámetros aprendibles.

Dentro de esta clase de métodos, Pourchot y Sigaud (2019) y Khadka y Tumer (2018) resuelven problemas de control físico-continuo, como el péndulo invertido o el róver lunar, con un controlador de red neuronal fungible. Sus enfoques son similares a la neuroevolución (Stanley y Miikkulainen, 2002; Floreano, Dürr, y Mattiussi, 2008; Lüders, Schläger, Korach, y Risi, 2017; Risi y Togelius, 2017), ya que ambas técnicas se basan en el uso de gradientes de política deterministas profundos (DDPG, según su sigla en inglés) (Lillicrap y cols., 2016) y utilizan un grupo compartido de actores entre los componentes de PG y RL.

Khadka y Tumer (2018) introdujeron el método *Evolutionary Reinforcement Learning* (ERL), un algoritmo híbrido que combina el DRL con AEs. En ERL, una población generada por un AE proporciona datos diversificados para entrenar a un agente RL, mientras que este agente se reinserta periódicamente en la población evolutiva para incorporar información basada en gradientes. Para coordinar este proceso, ERL utiliza un *buffer* de reproducción cíclico que almacena muestras generadas tanto por el enfoque evolutivo como por RL. Este *buffer* único se emplea para guiar pasos del AE o para refinar el modelo entrenado mediante RL. La combinación de AEs y DRL permite a ERL aprovechar la capacidad de exploración diversa y la estabilidad de un enfoque basado en poblaciones, mientras mantiene la eficiencia de aprendizaje y el uso de gradientes característicos del DRL. Este método ha demostrado un rendimiento significativamente superior en comparación con métodos previos de AEs y DRL en problemas de control continuo (Khadka y Tumer, 2018).

Pourchot y Sigaud (2019) introducen CEM-RL que resuelve la misma familia de problemas de manera similar a la presentada en el párrafo anterior. La diferencia más notable es que la población compartida no alimenta directamente a un algoritmo de PG, sino que se usa para actualizar una distribución de la cual se extrae una población.

Más relacionado con nuestro trabajo, Ahn, Kim, Lee, y Shin (2020) desarrollan el aprendizaje guiado por expertos genéticos (GEGL, según su sigla en inglés). En GEGL, las muestras se producen utilizando un generador de secuencias basado en RNN estocástico y se agregan a una cola de prioridad de

recompensa máxima (MRPQ, según su sigla en inglés). Un componente de PG aplica mutación y/o cruzamiento en cada elemento de la MRPQ. Particularmente, solo se aplica una generación de operadores evolutivos a cada muestra en la MRPQ. Como consecuencia, no hay noción de un operador de selección.

En contraste, en el trabajo de Mundhenk y cols. (2021), donde se presenta el modelo DSO, se concluye que tanto el operador de selección como la realización de múltiples generaciones de evolución son elementos cruciales para maximizar los beneficios de la PG. Después de realizar una generación de la PG, la población resultante se agrega a una segunda MRPQ. Las muestras de la unión de la MRPQ, basada en PG, y la MRPQ basada en RNN, se utilizan para entrenar la RNN. En contraste, los autores encuentran que las colas de prioridad no son necesarias y pueden impedir una exploración suficiente. Al igual que en los mencionados métodos ERL y CEM-RL, los pasos evolutivos y de entrenamiento de GEGL son uno a uno, lo que significa que cada paso de entrenamiento es seguido por exactamente una generación de PG. Además, PG utiliza una memoria persistente de poblaciones de muestras, la cual comparte con el componente de red neuronal. Este fuerte acoplamiento hace que el componente evolutivo sea mucho más interdependiente de la red neuronal.

3.2.5. Aplicación de algoritmos evolutivos en problemas de regresión simbólica multi-salida

En el contexto de este trabajo, es relevante mencionar el trabajo de J. Ferreira, Torres, y Pedemonte (2023), quienes desarrollaron el algoritmo $MOKP_{IM}$ (Multi-output Kaizen Programming Island Model) para resolver problemas de RS multi-salida. Aunque no utilizaron programación genética, su enfoque basado en AEs aborda problemas en ingeniería de procesos, incluyendo los casos de estudios presentados en la sección 4.4.4, utilizados en este trabajo para evaluar experimentalmente otros modelos. En particular, $MOKP_{IM}$ identifica términos compartidos entre las diferentes salidas mediante el intercambio de migrantes entre islas, lo que mejora la cooperación entre soluciones.

3.2.6. Incorporación de conocimientos previos en la regresión simbólica: restricciones en la imagen de la función

La integración de conocimientos previos en métodos de aprendizaje automático, incluyendo la programación genética, ha sido un tema de discusión en la literatura. Un trabajo relevante en este ámbito es el de Bladek y Krawiec (2019), quienes exploraron el uso de restricciones formales en combinación con la RS para incorporar conocimientos del dominio sobre la monotonía, convexidad o simetría de las funciones. Su enfoque, denominado *Programación Genética Basada en Contraejemplos*, utiliza *solvers* de teorías de satisfactibilidad de módulo (SMT, según su sigla en inglés) para verificar si los modelos candidatos de RS cumplen con las restricciones y para ampliar el conjunto de entrenamiento con

contraejemplos. Bladek y Krawiec observan que es muy improbable que la PG sintetice modelos que se ajusten a las restricciones.

En Versino, Tonda, y Bronkhorst (2017) investigaron específicamente la generación de modelos de esfuerzo de flujo para cobre utilizando PG, con y sin integración de conocimientos. Exploraron cuatro escenarios diferentes: (i) un *solver* de PG canónico que devuelve un frente de Pareto de calidad y complejidad del modelo, (ii) integración de conocimientos mediante variables transformadas, ponderación de muestras, introducción de puntos artificiales y soluciones iniciales, (iii) búsqueda de la derivada de la expresión para luego reconstruir la ecuación original mediante integración, y (iv) búsqueda de derivadas con puntos de datos artificiales adicionales. Este trabajo enfatizó la importancia de integrar conocimientos previos en el proceso de ajuste del modelo. Los autores observaron que el *solver* de PG canónico presentó un comportamiento impredecible respecto a las restricciones físicas; por lo tanto, el conjunto de datos muestreados por sí solo era insuficiente para guiar al *solver* hacia un modelo con bajo error y factibilidad física. Todas las técnicas probadas en Versino y cols. (2017) son optimistas y podrían producir soluciones inviables. Además, la adición de puntos de datos artificiales es subjetiva y no escala a problemas de alta dimensión según Kronberger y cols. (2021).

M. D. Schmidt y Lipson (2009) investigaron la influencia de integrar conocimientos expertos en el proceso de búsqueda evolutiva mediante un procedimiento denominado *seeding*. Primero, los autores generaron soluciones aproximadas resolviendo un problema más simple o encontrando una solución aproximada a un problema más complejo. Estas soluciones se utilizaron durante el procedimiento de *seeding* al insertar aproximaciones en la población inicial, en expresiones barajadas y en bloques constructivos. Los autores encontraron que el *seeding* mejora significativamente la convergencia y el rendimiento promedio de la aptitud en comparación con la ausencia de *seeding*. Entre los procedimientos de *seeding*, el *seeding* de bloques constructivos fue el más exitoso, permitiendo una convergencia más rápida incluso en problemas más complejos. Sin embargo, el *seeding* es un enfoque optimista y no garantiza que la solución final de RS se ajuste al conocimiento previo, según los autores.

Stewart y Ermon (2017) investigaron cómo utilizar el conocimiento previo en redes neuronales artificiales. Específicamente, incorporaron conocimientos físicos en la detección y el seguimiento de movimientos, así como relaciones causales en la detección de objetos. Aunque los experimentos fueron a pequeña escala, mostraron resultados prometedores y abrieron la posibilidad de entrenar una red neuronal artificial describiendo únicamente las propiedades de la función de aproximación. Esta idea es similar a la de *shape constraints*, que se menciona a continuación.

En Kronberger y cols. (2021), se investigó la adición de restricciones en la imagen de la función y sus derivadas para la incorporación de conocimiento previo en la RS, denominada *shape-constrained symbolic regression*. Este enfoque permite imponer ciertas propiedades deseadas a un modelo de regresión (o clasificación), por ejemplo, la monotonía de la función sobre entradas seleccionadas. El objetivo era encontrar modelos que se ajusten al comportamiento esperado

y que posean mejores capacidades de extrapolación. Los autores demostraron la viabilidad de la idea. Además, propusieron y compararon dos AEs para la *shape-constrained symbolic regression*: una extensión de la PG basada en árboles que descarta soluciones inviables en el paso de selección, y un AE de dos poblaciones que separa las soluciones viables de las inviables. En ambos algoritmos se utilizó aritmética de intervalos para aproximar los límites de los modelos y sus derivadas parciales. Los algoritmos fueron probados en un conjunto de 19 problemas de regresión sintéticos y cuatro problemas del mundo real. Ambos algoritmos lograron identificar modelos que cumplen con las *shape constraints*, lo cual no ocurre con los algoritmos de RS no modificados. Sin embargo, la precisión predictiva de los modelos con restricciones fue inferior tanto en el conjunto de entrenamiento como en el de validación.

Recientemente, se ha descrito un enfoque para la integración de conocimiento en la RS con una motivación similar a la descrita en el párrafo anterior (L. Li, Fan, Singh, y Riley, 2019). En lugar de *shape constraints*, los autores utilizan *priors semánticos* en forma de potencias para las expresiones simbólicas. Li y otros autores describen un algoritmo de Búsqueda de Árbol de Monte Carlo Guiada por Redes Neuronales (MCTS, según su sigla en inglés) para buscar modelos de RS que se ajusten al conocimiento previo. Los autores emplean una gramática libre de contexto para generar expresiones simbólicas, y utilizan MCTS para generar soluciones, donde una red neuronal predice la siguiente regla de producción dada una secuencia de reglas ya aplicadas y las restricciones de potencia principal. Dicho algoritmo se comparó con variaciones de MCTS y una implementación de PG utilizando el *framework DEAP*. Los resultados de tal trabajo muestran un rendimiento superior del MCTS guiado por redes neuronales con una alta tasa de éxito en casos más fáciles, pero tasas mucho más bajas en casos más difíciles, aunque aún superiores a las tasas de éxito de PG.

Uno de los objetivos del mencionado trabajo de Kronberger y cols. (2021) es mejorar la extrapolación de las soluciones de RS. Similar a la regresión polinómica, los modelos de RS podrían producir valores extremos al extrapolar. Curiosamente, el comportamiento de extrapolación de los modelos de RS ha sido en gran medida ignorado en la literatura, con pocas excepciones. F. A. Castillo, Villa, y Kordon (2013) investigaron la diferencia en la variación de respuesta de diferentes modelos generados por PG para tres conjuntos de datos distintos. Mostraron que, en algunas situaciones, un algoritmo de PG puede generar modelos con comportamientos diferentes mientras presenta un valor similar de R^2. Estos resultados motivan la necesidad de un análisis posterior de la validez del modelo con respecto al sistema estudiado y una verificación de qué tan bien extrapolan tales modelos.

La capacidad de extrapolación de PG también había sido estudiada por Castillo (F. Castillo, Marshall, Green, y Kordon, 2003). En el anterior trabajo, los autores evaluaron y compararon la mejor solución obtenida por un algoritmo de PG con un modelo lineal que utiliza las variables transformadas encontradas por el modelo PG. Los experimentos sugieren que los modelos de PG tienen buenas capacidades de interpolación pero con un error de extrapolación moderado. El

enfoque propuesto presentó buenas capacidades de interpolación y extrapolación, sugiriendo que, al menos, los modelos de PG pueden guiar el proceso de construcción de un nuevo espacio transformado de variables.

Finalmente, en el ya mencionado trabajo de Stewart y Ermon (2017), los autores entrenaron una red neuronal introduciendo conocimiento previo mediante un término de penalización en la función de pérdida, de modo que el modelo generado es consistente con un comportamiento físico, produciendo un modelo de caja negra de los datos.

3.2.7. Operador de selección de Deb para el manejo de restricciones

Como ya se ha mencionado en varios trabajos previos, en problemas de optimización del mundo real, es común que existan restricciones de desigualdad e igualdad, lo que convierte a estos problemas en desafíos de optimización bajo un conjunto de restricciones. Los métodos más populares para manejar restricciones en algoritmos genéticos y métodos clásicos de optimización han sido las funciones de penalización debido a su simplicidad. Sin embargo, estos métodos requieren establecer parámetros de penalización adecuados, lo cual implica realizar pruebas experimentales para encontrar el valor óptimo, y no siempre garantizan un rendimiento satisfactorio.

El enfoque presentado por Deb (2000) elimina la necesidad de parámetros de penalización, utilizando la selección por torneo en los algoritmos genéticos. Al realizar comparaciones de a pares entre soluciones, se propone un método que maneja restricciones sin parámetros adicionales, aprovechando la naturaleza poblacional de los algoritmos.

El método se basa en las siguientes reglas para la selección por torneo: cuando se comparan dos soluciones factibles, se elige aquella con mejor valor de la función objetivo. Si se compara una solución factible con una solución no factible, se elige la solución factible. En el caso de que ambas soluciones sean no factibles, se elige aquella con menor violación de restricciones.

El anterior enfoque tiene como objetivo que las soluciones no factibles sean guiadas hacia la región factible, y que las soluciones factibles sean priorizadas durante el proceso de selección. Aunque existen otras implementaciones, como la de los trabajos de Michalewicz (1995), D. Powell y Skolnick (1993) y Richardson, Palmer, Liepins, y Hilliard (1989), en las que se imponen criterios similares a los anteriores en sus enfoques de gestión de restricciones, todas estas implementaciones utilizaban diferentes medidas de violación de restricciones que seguían necesitando un parámetro de penalización para cada restricción (Deb, 2000).

Por otro lado, en el artículo mencionado, Deb propone un método que, además de utilizar el operador de selección descrito anteriormente, introduce un esquema de nicho una vez que se ha encontrado un número adecuado de soluciones factibles. Este esquema de nicho permite preservar la diversidad dentro de la población, lo que facilita que el operador de cruzamiento continúe explorando y encontrando soluciones factibles de mayor calidad a lo largo de las generaciones.

En métodos convencionales, las funciones de penalización requieren parámetros específicos para equilibrar el valor de la función objetivo con la magnitud de la violación de restricciones. En contraste, en el método propuesto, las soluciones no se comparan simultáneamente en términos de la función objetivo y la violación de restricciones. Las *decisiones* se toman basándose únicamente en la factibilidad de las soluciones y, para las soluciones no factibles, únicamente en las violaciones de restricciones. Esto elimina la necesidad de calcular parámetros de penalización y hace que el método sea más robusto y eficiente.

Los resultados reportados en el artículo indican varias ventajas del método propuesto. En primer lugar, se evita la necesidad de experimentar con parámetros de penalización, simplificando el proceso de ajuste del algoritmo. En segundo lugar, el uso del esquema de nicho permite mantener la diversidad entre soluciones factibles, facilitando que el operador de cruzamiento encuentre mejores soluciones en cada generación. En pruebas realizadas, incluyendo un problema de diseño de ingeniería, el algoritmo con el manejo de restricciones que descrito anteriormente, encontró soluciones más cercanas al óptimo verdadero en comparación con métodos anteriores, e incluso superó las mejores soluciones previamente reportadas (Deb, 2000)

Deb concluyó que los resultados del estudio muestran que el manejo de restricciones propuesto es prometedor para realizar tareas de optimización restringida de manera confiable y eficiente mediante el uso de algoritmos genéticos.

3.3. Conclusiones de la revisión

Dada la abundancia de referencias encontradas y la calidad de los resultados reportados, se puede concluir que la programación genética, las redes neuronales y los enfoques híbridos son opciones destacadas para la generación de modelos subrogados. Estas técnicas han demostrado ser adecuadas para enfrentar los desafíos de modelado en problemas reales y *benchmarks* en el ámbito de la ingeniería de procesos.

En particular, los enfoques que combinan PG con DL, como el caso de DSO, y aquellos que emplean técnicas de mínimos cuadrados para optimizar los modelos simbólicos, como Operon que fue mencionado en el capítulo anterior, han mostrado un gran potencial. Además, se ha resaltado la importancia de considerar métodos multiobjetivo para abordar criterios adicionales como *shape constraints* y puntos indefinidos en los datos de validación. Por otro lado, se vió que para el manejo de restricciones, existen técnicas prometedoras en el contexto monoobjetivo que eliminan la necesidad de ajustar parámetros de penalización, simplificando así el proceso de optimización y mejorando la eficiencia del algoritmo.

Capítulo 4

Selección, implementación y ajuste de los métodos estudiados para la evaluación experimental

A partir del relevamiento realizado en las secciones 2 y 3, se utilizarán tres enfoques distintos para trabajar sobre el problema de la RS.

El primer enfoque consiste en considerar solamente la minimización del error de la función generada por los métodos. Dicha aproximación, que denominaremos *enfoque monoobjetivo*, se implementó empleando tres herramientas: DSO, DSR y Operon. Estos algoritmos fueron seleccionados no solo por sus capacidades diferenciadas y enfoques prometedores en la RS, sino también porque están disponibles en *frameworks* de código abierto, lo que permite realizar las modificaciones necesarias para adaptar los modelos a nuestras necesidades específicas. Además, durante el proceso fue necesario ajustar y corregir dichos *frameworks* para asegurar una comparación adecuada entre estos algoritmos, lo cual se detalla a lo largo de este capítulo.

El segundo enfoque toma ideas de trabajos previos, similares a las de Kronberger y cols. (2021) y transforma el problema en uno multiobjetivo donde además de minimizar el error, se agrega un segundo objetivo "estructural"sobre la función que se quiere obtener. Por ejemplo, la presencia de valores indefinidos, infinitos o la necesidad de mantener ciertas propiedades, como la monotonicidad de la función en algunas de sus variables. Para lograrlo, se seleccionó uno de los métodos estudiados en el enfoque monoobjetivo y se lo adaptó para que pudiera operar bajo un criterio multiobjetivo.

Finalmente, y a partir del relevamiento realizado, se identificó el operador de selección propuesto por Deb como una alternativa interesante. Dicho operador fue planteado para lidiar con el manejo de restricciones en AE y el caso de las

restricciones "estructurales" pueden considerarse como un caso particular. A partir de dicho operador, se implementó una variante de DSO que utiliza este criterio de selección.

En este capítulo se describen también las configuraciones utilizadas, incluyendo los conjuntos de datos y las funciones empleadas como *benchmarks*. Estos *benchmarks* permiten validar las técnicas propuestas y analizar su aplicabilidad en escenarios prácticos, asegurando así que los enfoques monoobjetivo, multiobjetivo y la variante de DSO con el operador de Deb sean comparados de forma sólida y representativa.

4.1. Metodología para preparar los *frameworks* para la evaluación experimental siguiendo un enfoque monoobjetivo

Para poder llevar a cabo el análisis experimental siguiendo este enfoque, se decidió realizar un proceso preliminar de evaluación para algunos problemas del *benchmark* Vladislavleva (ver Sección 4.4.3), con el fin de corroborar que en cada fase de los experimentos se pudiese contar con los datos necesarios para iniciarlos, y que se generasen los resultados esperados. De esta forma se detectan una serie de *bugs* tanto en el *framework* DSO, donde se ejecutan el anterior algoritmo y DSR, como la ausencia de algunas herramientas para el reporte de resultados en Operon.

4.1.1. Instalación y ambiente

Como se mencionó en los capítulos anteriores, los algoritmos evaluados incluyen implementaciones en Python para DSO y DSR, que utilizan el *framework DEAP* para las operaciones de PG y TensorFlow para la RNN. Por su parte, Operon está implementado en C++, pero se integró a nuestro flujo de trabajo mediante `pyoperon`, una interfaz compatible con `sklearn`, lo que permitió ejecutar dicho algoritmo desde Python.

Para facilitar la portabilidad y la reproducción conjunta y reproducible de los experimentos, se utilizaron *DevContainers* en Docker[1], cuyo entorno estaba basado en una imagen de `Ubuntu 18.04`. Se confeccionaron archivos Dockerfile que permitieron no solo facilitar dicha instalación, sino que también aumentar la velocidad y aportar comodidad a la hora de escribir código, ya sea en el ambiente de DSO para realizar modificaciones, como en Operon para implementar las funciones auxiliares necesarias. Dentro de este entorno, se configuraron todas las dependencias necesarias, tanto para DSO como para Operon. Cada contenedor incluyó configuraciones para Python, así como bibliotecas y herramientas auxiliares como `git`, `software-properties-common`, y versiones específicas de `pip` y `setuptools`. Además, el directorio de trabajo en el contenedor fue confi-

[1] https://www.docker.com/

gurado para manejar de forma adecuada el código fuente, los datos y resultados, utilizando volúmenes para la persistencia.

A continuación, se describe el entorno y las máquinas en las que se llevaron a cabo las ejecuciones:

1. **Computadora 1**: Se utilizó una máquina con un procesador **AMD Ryzen 5 7530U** de 6 núcleos a 2.00 GHz, equipada con **16 GB de RAM**, corriendo **Windows 11 Pro**. Como mencionamos, se utiliza Docker para ejecutar el contenedor Ubuntu.

2. **Computadora 2**: Una segunda máquina con un procesador **Intel i3-1215U** de 8 núcleos a 1.20 GHz, con **16 GB de RAM**, ejecutando **Windows 11 Home 23H2**. Al igual que en la primera computadora, Docker se utilizó para ejecutar los DevContainers con configuraciones idénticas a las de la primera máquina.

Cabe destacar que esta plataforma de ejecución se utilizará también en los experimentos siguiendo otros enfoques.

El uso de Docker permitió estandarizar el entorno de pruebas, independientemente del sistema operativo *host*, y asegurar que las versiones de bibliotecas y dependencias fueran consistentes en todas las ejecuciones. Asimismo, se aprovechó la capacidad de ejecutar múltiples experimentos en paralelo, lo cual fue particularmente beneficioso en el caso de DSO, donde las ejecuciones paralelas permitían reducir el tiempo total de los experimentos.

4.1.2. Modificaciones al *framework* DSO

A continuación, se describen los distintos *bugs* encontrados al *framework* DSO, con las correspondientes correcciones realizadas para poder llevar a cabo los experimentos.

4.1.2.1. Generación de los *datasets*

Durante la generación de los *datasets* que serían utilizados en los algoritmos, se presentaron varios desafíos relacionados con la creación de variables múltiples $(x_1, x_2, ..., x_n)$ basadas en especificaciones de rango, paso y distribución. Estos problemas incluían:

- **Manejo de especificaciones `all`:** Las especificaciones que utilizaban la palabra clave `all`, que indicaba que todas las variables implicadas abarcaban el mismo rango y cantidad de puntos a seleccionar, no se estaban aplicando correctamente a todas las variables. Esto generaba configuraciones de *datasets* incorrectas.

- **Cálculo incorrecto del tamaño del *dataset*:** El cálculo del tamaño del *dataset* no consideraba correctamente el número total de combinaciones posibles entre variables, especialmente al usar distribuciones equiespaciadas (E), lo que llevaba a un tamaño incorrecto del *dataset*.

- **Errores de precisión en cálculos de pasos:** Surgieron errores de precisión al calcular la cantidad de puntos basados en un rango y paso dado, debido a la naturaleza de la representación de números en punto flotante de Python, lo que resultó en tamaños incorrectos para el *dataset*.

Para resolver estos problemas, se realizaron las siguientes modificaciones:

- **Actualización del manejo de `all`:** Se ajustó la función de generación del *dataset* para que, cuando se utilizara la especificación `all`, el tamaño calculado para una variable se aplicara correctamente a todas, elevando el tamaño por variable al número de variables.

- **Corrección en el cálculo del tamaño del *dataset*:** Se revisó el cálculo de tamaños para garantizar que las combinaciones posibles entre las variables fueran correctamente multiplicadas, obteniendo así un tamaño preciso del *dataset*.

- **Ajuste en la precisión de los cálculos de pasos:** Se implementaron técnicas para manejar la precisión de los números representados en punto flotante, asegurando que el tamaño del *dataset* reflejara con exactitud los rangos y pasos especificados, eliminando errores de precisión.

Después de estos ajustes, la generación del *dataset* comenzó a funcionar correctamente, generando los puntos esperados para cada variable según las especificaciones, incluyendo los casos donde se utilizaba `all` para definir un rango común para todas las variables. Además, se resolvieron los errores de precisión en los cálculos.

4.1.2.2. Reporte de resultados

Se detectó un problema en la generación del *file summary*, un archivo destinado a recopilar las estadísticas y métricas más relevantes derivadas de un conjunto de ejecuciones independientes. Para algunos problemas del *benchmark* Vladislavleva que se verán más adelante, se observó que la métrica RMSE no se reportaba para ciertas funciones resultantes, lo cual afectaba la completitud de los resultados.

Este problema ocurrió porque algunas expresiones generadas por DSO incluían operaciones que, bajo ciertas condiciones, provocaban errores de evaluación. Esto resultaba en valores `NaN` (*Not a Number*) o infinitos, lo que impedía el cálculo adecuado del *RMSE*. Como consecuencia, en las ejecuciones independientes afectadas, dicha métrica no se registraba en el archivo resumen, comprometiendo la representación numérica de los resultados y dificultando el posterior análisis.

Para resolver este problema, se implementó un conjunto de funciones en Python que permitieron completar los valores faltantes de RMSE en el archivo de resumen de la siguiente manera:

- **Evaluación segura de expresiones**: Se desarrolló la función `eval_expression()` que realiza una evaluación segura de las expresiones generadas. Esta función trata de evaluar cada expresión en el conjunto de datos de validación, capturando excepciones y manejando los casos donde las operaciones producen valores `NaN` o infinitos.

- **Reemplazo de funciones matemáticas**: Se implementó la función `replace_functions()` que reemplaza funciones matemáticas como `sin()`, `cos()`, `exp()`, entre otras, por sus equivalentes seguros de `NumPy`.

- **Cálculo y actualización de RMSE**: Se añadió lógica para revisar el archivo de resumen. Cuando se encuentra un valor faltante de RMSE, se recalcula utilizando la función `calculate_neg_rmse()` (*neg* para ser consistentes con las nomenclaturas del *framework*, donde varias métricas se reportaban con valores negativos), que evalúa la expresión de la función sobre los datos de validación y, si es posible, calcula el *RMSE*. Si la expresión original no es válida, se intenta recuperar el *RMSE* de la mejor expresión disponible en el archivo *hall of fame* (HOF) correspondiente a la ejecución independiente con dicho problema. Es decir, la siguiente expresión que obtuvo mejor valor para la función objetivo sobre los datos de entrenamiento.

Tras la implementación de estas soluciones, se logró completar correctamente los valores faltantes del error en el archivo de resumen, asegurando así que todas las ejecuciones registraran adecuadamente el RMSE.

4.1.2.3. Interfaz de Sklearn no soporta uso del componente de PG

El problema se encuentra en la función `gp_meld`, la cual, al momento de la redacción de este informe, no es compatible con la interfaz de `sklearn`. Este problema no solo fue reportado por un usuario en GitHub[2], sino que también se brindaba un `warning` en el archivo `sklearn.py`. En dicho archivo se incluía un comentario que indicaba que existe una tarea pendiente (`TBD: To Be Done`) para agregar soporte a la función `gp_meld` dentro de la interfaz de `sklearn`.

Actualmente, la función `gp_meld` depende de objetos `BenchmarkDataset`, que son estructuras específicas utilizadas dentro del *framework* para manejar conjuntos de datos de *benchmarking*. Sin embargo, la interfaz de `sklearn` usa un formato de datos estándar `(X, y)`, donde `X` representa las características de entrada e `y` las etiquetas o salidas correspondientes.

Para solucionar este problema, se creó una variante del archivo `run.py`. Dicho archivo es un *script* de lanzamiento paralelo y de un solo punto diseñado para ejecutar DSO o DSR en un conjunto de *benchmarks*, asegurando así la funcionalidad esperada del componente de PG. De esta forma se obtienen los mismos resultados que al ejecutar comandos vía terminal.

[2] https://github.com/dso-org/deep-symbolic-optimization/issues/94

4.1.3. Modificaciones al *framework* Operon

Operon, como se señaló previamente en la Sección 2.3.2, es un *framework* de C++ que cuenta con una implementación de una interfaz en Python llamada *pyoperon* mediante la biblioteca `scikit-learn` que permite acceder desde Python a los métodos programados en C++. Más específicamente, se destacan de la interfaz las siguientes operaciones:

1. `model.__init__(config)`: Se permite inicializar el modelo con los parámetros necesarios mediante la creación del objeto utilizando este método. El parámetro `config` es un archivo JSON que contiene los pares clave-valor de los parámetros a utilizar.

2. `model.train(X,y)`: Este método sirve para entrenar el modelo y ajustarlo a los valores pasados por parámetro, dichos valores `X` e `y` son vectores de *numpy* de dimensión $D \times n$ y $D \times 1$ respectivamente, donde `X` son los valores de las variables e `y` es la salida de la función objetivo obtenida al sustituir por dichas variables. Por lo tanto, n representa la cantidad de variables de las expresiones y D el largo del *dataset* utilizado para entrenar.

3. `model.predict(X)`: Siendo `X` un vector con las mismas características que en el ítem anterior, esta función permite inferir a partir de los datos de entrada, obteniendo una salida `y` predicha por el modelo.

4. `model.get_model_string()`: Retorna el modelo en formato simbólico obtenido luego del entrenamiento. A efectos de mejorar la interpretabilidad y dado que este método genera expresiones extensas, el *string* retornado es utilizado como entrada por la biblioteca `SymPy` para realizar una reducción y simplificación matemática de la expresión. Esto permite identificar y eliminar términos redundantes o irrelevantes, mejorando la interpretabilidad del modelo sin afectar su precisión.

4.1.3.1. Reporte de RMSE

Al realizar las inferencias, `pyoperon` no proporciona directamente herramientas en la interfaz de `sklearn` para obtener métricas de RMSE. Por lo tanto, se implementa una solución utilizando la biblioteca `pandas`, que permite entrenar los modelos y, tras cada entrenamiento, guardar tanto el modelo mediante `model.get_model_string()` (el modelo fuente y el modelo simplificado mediante `SymPy`) como las métricas calculadas, incluido el tiempo de entrenamiento. Se decide generar un archivo resumen con las métricas más relevantes obtenidas tras las ejecuciones independientes, de manera similar a lo que hace DSO. Sin embargo, al igual que con DSO, en algunas ejecuciones se generaron valores de RMSE no válidos debido a que la expresión obtenida mediante la RS no estaba definida en ciertos puntos del conjunto de validación. Por lo tanto, se procedió de igual forma que con DSO, recuperando el valor de RMSE en validación de las siguientes expresiones generadas en la ejecución, que brindaron menores errores para el conjunto de entrenamiento.

4.2. Enfoque multiobjetivo

A partir de los buenos resultados relevados en la bibliografía de DSO, motivó la decisión de trabajar con este algoritmo en el contexto del enfoque multiobjetivo. La principal ventaja de DSO, además de sus buenos resultados, radica en su componente de aprendizaje profundo, el cual no está presente en Operon y le confiere a DSO un potencial adicional para abordar problemas complejos. Dado que DSO no cuenta con un componente multiobjetivo, se debió realizar una extensión que permitiera trabajar con este tipo de problemas.

4.2.1. Integración de DSO con NSGA-II para permitir un enfoque multiobjetivo

En el caso de DSO, el *framework* no provee una implementación de NSGA-II (ni otro algoritmo multiobjetivo), por lo que se debió implementar por separado. Para llevar a cabo dicha implementación, se utiliza el *framework* jMetalPy (descrito en la Sección 2.3.5), el cual provee una implementación de NSGA-II. Se detallará la estructura del algoritmo y como se adaptó a DSO para soportar o incorporar un enfoque multiobjetivo. Finalmente se explicará las consideraciones a nivel de optimización que se tuvieron en cuenta a la hora de integrar dichos sistemas.

4.2.1.1. Características de la implementación de NSGA-II en jMetalPy

La clase `NSGAII` en la implementación de jMetalPy representa el algoritmo genético multiobjetivo NSGA-II. El algoritmo hereda la clase `GeneticAlgorithm` (ver Sección 2.3.5.1) que implementa los operadores clásicos de PG, y se distingue principalmente por su función de reemplazo característica.

Durante su inicialización, se definen parámetros esenciales como el problema a resolver, el tamaño de la población, los operadores de mutación y cruzamiento, y los criterios para la selección y finalización.

Para la selección de soluciones, la implementación utiliza la combinación de dos operadores: el *fast non-dominated sorting* (`FastNonDominatedRanking`) y la *crowding distance* (`CrowdingDistance`). El mecanismo de reemplazo consiste en combinar la población actual con la descendencia generada en cada iteración. Luego, aplicando el *ranking* y la estimación de densidad, se seleccionan las mejores soluciones para formar la nueva población, garantizando que las de mayor *fitness* sobrevivan.

En cuanto a la finalización del proceso, la implementación es flexible, permitiendo configurar distintos criterios de terminación y evaluadores.

Además, el algoritmo puede operar en modo *steady-state*, lo que significa que se genera un solo descendiente por generación. Este enfoque es útil cuando es necesario evaluar nuevas soluciones de manera continua, sin reemplazar grandes partes de la población de una sola vez.

Finalmente, la implementación incluye la posibilidad de configurar un comparador de dominancia, lo que ofrece un mayor control sobre la manera en que se evalúan las soluciones en términos de su dominancia relativa.

4.2.1.2. Estructura

Para incorporar el algoritmo NSGA-II a DSO, se opta por integrar dicho algoritmo multiobjetivo al componente de PG descrito en la sección 2.5.3.

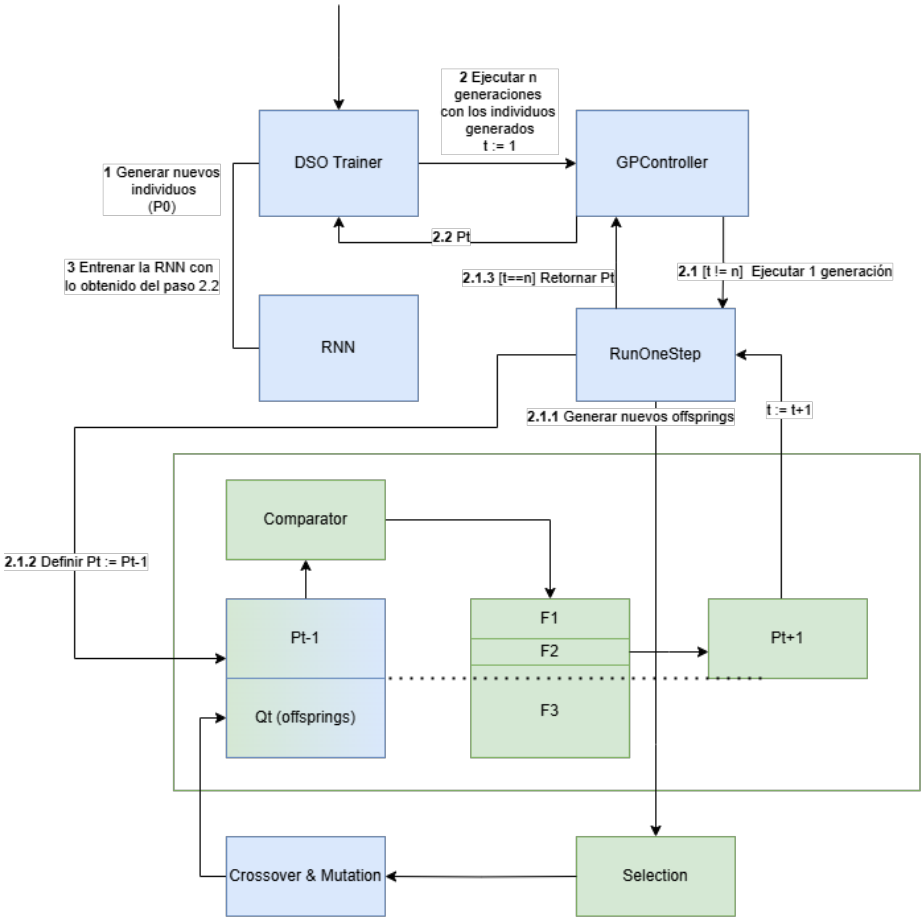

Figura 4.1: Integración de DSO y jMetalPy para un enfoque multiobjetivo.

Como se puede observar en la Figura 4.1, el sistema en cuestión cuenta con la implementación provista por DSO (representada en azul), la cual está compuesta por una clase `Trainer` que maneja tanto la red neuronal a entrenar, como las invocaciones a la clase `GPController`. Esta última clase se encarga de procesar las n generaciones de PG, utilizando como población inicial la retornada

por la red neuronal mediante el *framework DEAP* ya mencionado. En cada generación de PG, se toma la población generada por la RNN, definiéndola como P_0 y se le aplican los operadores que DSO ya utiliza con normalidad para generar la población P_1 a excepción de la selección de individuos, ya que la misma se modifica para adaptarla al algoritmo NSGA-II. Como se puede ver en la Figura 4.1, se utiliza (en verde y por separado) la selección binaria adaptada a NSGA-II y luego los operadores de cruzamiento y mutación de DSO. Posteriormente, ambas poblaciones se introducen al algoritmo NSGA-II provisto por jMetalPy, retornando una población de la misma cantidad de individuos que P_1, a esta población la llamaremos P_2. En este punto se obtienen los *offsprings* que definen la siguiente generación, se retorna entonces al bucle de DSO, en el que se pregunta si el número de generaciones es el solicitado por iteración. En caso de que este no supere el umbral máximo de generaciones definido previamente, se repite el paso desde el punto 2.1.2 de la Figura 4.1. De esta forma, se utilizan nuevamente los operadores de DSO para generar una nueva población de *offsprings* y posteriormente aplicarlo a NSGA-II para obtener la población P_3. Este proceso se repite hasta que se alcance el máximo número de generaciones.

En ese caso, se retorna a `DSOTrainer` la población generada para que esta entrene la red neuronal y produzca los individuos de la siguiente iteración. Nótese que el término aquí no es generación, pues se comienza una nueva iteración de DSO como algoritmo completo, incluyendo el ajuste de la red neuronal, por lo tanto, el t (ver Figura 4.1, parte 2.2) que marca la generación actual, se reinicia a cero tras esto.

4.2.2. *Shape constraints* consideradas

Como se mencionó previamente, las *shape constraints* (restricciones de forma) pueden ser utilizadas para imponer ciertas propiedades deseadas en los modelos de RS generados. En particular, se pondrá foco en las propiedades de monotonicidad y no-negatividad, que son cruciales para garantizar comportamientos predecibles en los modelos. Estas propiedades se definen matemáticamente en la Tabla 4.1.

Propiedad	**Formulación matemática**
No-negatividad	$f(x) \geq 0$
Monotonía no decreciente	$\frac{\partial}{\partial x_i} f(x) \geq 0$
Monotonía no creciente	$\frac{\partial}{\partial x_i} f(x) \leq 0$

Tabla 4.1: *Shape constraints* utilizadas en los algoritmos de RS. Todas las restricciones asumen un dominio $l_i \leq x_i \leq u_i$ para cada variable x_i.

4.2.3. Tratamiento de puntos indefinidos en las evaluaciones

Además de las *shape constraints*, es crucial considerar los puntos indefinidos que pueden surgir en las funciones generadas por los modelos. En el Capítulo 5, se verá que hay situaciones donde las expresiones obtenidas por los algoritmos presentan problemas como divisiones por cero, raíces de números negativos o logaritmos de valores no válidos, lo que causa que las funciones no estén definidas en ciertas regiones del dominio. Asimismo, cuando los modelos generan valores negativos en situaciones donde se espera que sean no negativos (como en algunas restricciones del problema), se consideran estos puntos en el dominio como indefinidos y serán computados como tales a la hora de contar dichos puntos. Esto debido a que este tipo de comportamiento es igualmente perjudicial para la calidad y estabilidad de los modelos.

Para abordar este problema, se evaluarán los modelos considerando como objetivos a minimizar tanto el *RMSE* como la cantidad de puntos indefinidos.

El objetivo es minimizar ambos factores simultáneamente en los datos de entrenamiento, para obtener modelos que no solo se ajusten correctamente a los datos, sino que también sean válidos en todo el dominio y respeten las restricciones impuestas por el problema.

4.3. Enfoque monoobjetivo usando el operador de selección de Deb

En esta sección se presenta la incorporación del operador de selección propuesto por Deb (2000), a una variante del algoritmo DSO. La estrategia para el operador presentado por Deb, introducida en la Sección 3.2.7, propone un operador de selección que permite manejar restricciones con prioridades en los modelos. Esto se integra en este proyecto a la hora de ejecutar el componente de PG de DSO.

4.3.1. Características de la implementación

Para integrar el operador mencionado en DSO, se destacaron dos soluciones clave que facilitaron la construcción del algoritmo resultante, aprovechando componentes existentes para simplificar el proceso.

En primer lugar, se aprovechó la capacidad del algoritmo DSO-MO para convertir los individuos de *DEAP*, utilizados en la versión original de DSO, a individuos de jMetalPy. Este proceso incluye el cálculo de toda la información necesaria para realizar la selección, como el error, el cumplimiento de las *shape constraints* o la cantidad de puntos indefinidos para los datos de entrenamiento, registrados en cada solución. Como se había mencionado, al ser almacenados los individuos en caché para ser reutilizados en el caso que prevalezcan en futuras generaciones, los valores no deben recalcularse al tratarse del mismo individuo y conjunto de datos.

En segundo lugar, se adaptó la selección por torneo binario proporcionada por jMetalPy para implementar la metodología de selección por torneo descrita por Deb (Deb, 2000).

Finalmente, el *offspring* resultante se pasa nuevamente a DSO, para que continúe con el ciclo de programación genética. Así, a pesar de que se utilizan componentes del DSO-MO, el operador mantiene un enfoque monoobjetivo. Esta adaptación permitió reducir los tiempos de ejecución al aprovechar la capacidad de *cachear* los individuos de jMetalPy, que contienen información relevante para el selector.

4.4. Benchmarks

El término *benchmark* proviene del inglés y la forma más adecuada de traducirlo según este contexto podría ser *punto de referencia*. Refiere a un estándar definido para medir y comparar el rendimiento de algo en particular. En RS, se trata de un conjunto de problemas estandarizados que se utilizan para evaluar y comparar el desempeño de diversos algoritmos, ya sea de PG, ML u otros enfoques relacionados.

En esta sección, se analizan los problemas específicos que aborda el proyecto, abarcando tanto problemas estándar de *benchmarking* como aplicaciones en casos reales. Además, se describen los conjuntos de datos provenientes de estos, seleccionados para validar los algoritmos, con el objetivo de analizar su aplicabilidad en escenarios prácticos.

4.4.1. SRBench

Generalmente, la investigación sobre RS ha tenido dificultades para evaluar y clasificar nuevos métodos de una manera que facilite su adopción (La Cava y cols., 2021). En el anterior trabajo citado se plantea la hipótesis de que esta deficiencia se debe, al menos en cierta medida, a la falta de *benchmarks* estandarizados, transparentes y reproducibles. El objetivo de SRBench es contrarrestar este efecto.

SRBench es un extenso *benchmark* que incluye, además de los problemas, distintos métodos de RS y otros de aprendizaje automático. Su objetivo es evaluar la precisión de las expresiones, simplicidad y la capacidad de los algoritmos para redescubrir la función generadora de datos en problemas físicos. Los resultados brindados por SRBench mostraron que los algoritmos de RS basados en PG tienden a ser los de mejor rendimiento actualmente en una variedad de problemas tabulares del mundo real. Además, se demostró que la RS es en sí misma un enfoque viable para modelar datos tabulares, ya que en muchos casos las expresiones descubiertas mostraron una precisión similar a los métodos de ML comúnmente aceptados como los mejores para esta tarea (de França y cols., 2023).

SRBench se caracteriza por ser un recurso de código abierto, ampliamente accesible para la comunidad de investigación. Realiza un exhaustivo relevamien-

to que incluye, además de los problemas de prueba, catorce métodos de RS y siete métodos de aprendizaje automático. SRBench incluye 252 conjuntos de datos del PMLB[3], los cuales están disponibles en formato *.tsv*. Estos conjuntos de datos abarcan tanto problemas con información previa (130 conjuntos de datos) como sin información previa (122 conjuntos de datos), cubriendo una amplia gama de problemas tanto sintéticos como del mundo real.

4.4.2. Funciones de Feynman

Uno de los conjuntos de datos más populares a la hora de evaluar algoritmos en problemas de RS, es el *dataset* de Feynman (Udrescu y Tegmark, 2020; Aldeia y de França, 2022). Este cuenta con más de 100 ecuaciones inspiradas en leyes físicas, sistemas físicos, originadas de *Feynman Lectures on Physics* (Feynman, Leighton, y Sands, 1964). Cada una de estas ecuaciones está contemplada en SRBench.

Las funciones genéricas $f(x_1, \ldots, x_n)$ suelen ser complicadas y casi imposibles de descubrir mediante RS. Sin embargo, las funciones que aparecen en la física y muchas otras aplicaciones científicas suelen tener algunas de las siguientes propiedades simplificadoras que facilitan su descubrimiento.

Es común que f y las variables de las que depende tengan unidades físicas conocidas, en otros casos, la función o partes de ella toman la forma de polinomios de bajo grado. Además, algunas veces se cumple que f es una composición de un conjunto pequeño de funciones elementales, cada una de ellas con no más de dos argumentos. Otra propiedad común es la suavidad de f, ya que muchas veces estas funciones son continuas o incluso analíticas en su dominio. También es frecuente que presenten algún tipo de simetría, ya sea de traslación, rotación o escala con respecto a alguna de sus variables. Finalmente, muchas de estas funciones son separables, lo que significa que pueden escribirse como la suma o el producto de dos partes que no comparten variables en común.

Aunque estas propiedades son comunes en muchas funciones científicas, el motivo por el que se presentan con tanta frecuencia sigue siendo un tema controvertido y no del todo comprendido, como han señalado varios autores (Poggio, Mhaskar, Rosasco, Miranda, y Liao, 2016; Lin, Tegmark, y Rolnick, 2017). Sin embargo, esto no ha impedido el desarrollo de algoritmos diseñados específicamente para explotar estas propiedades con el fin de facilitar la RS, como el ya mencionado AIFeynman, un método inspirado en la física (Feynman y cols., 1964).

Debido a razones de espacio, las tablas con la información de las funciones de Feynman se presentarán en la Sección A.1 del Anexo. En la Tabla A.1 se presentan los distintos conjuntos de funciones y terminales (*Function and Terminal Set*) que se utilizarán en los algoritmos para generar soluciones candidatas a los problemas de regresión. Posteriormente, en la Tabla A.2, se especifican los conjuntos de entrenamiento y validación, que fueron generados de manera independiente. Para la columna que define el espacio de entradas, $U[a, b]$ representa

[3] https://github.com/EpistasisLab/pmlb

muestras aleatorias uniformes extraídas del intervalo $[a, b]$, ambos extremos incluidos, con una cantidad ajustada de forma que se generen aproximadamente 10.000 puntos en total para cada conjunto. Asimismo, en dicha tabla se incluye para cada función del *dataset*, el conjunto de funciones y terminales a utilizar (*Function-Terminal Set*).

4.4.3. Funciones de Vladislavleva

Las ecuaciones de Vladislavleva se refieren a un conjunto de 8 funciones comúnmente utilizadas en el campo de la RS y que no están incluídas en SRBench. Fueron propuestas por Ekaterina Vladislavleva en su trabajo para evaluar el desempeño de algoritmos de RS (Vladislavleva, Smits, y den Hertog, 2009). Estas ecuaciones varían en complejidad y se utilizan para probar la capacidad de un método para descubrir relaciones no lineales en datos.

En la Tabla 4.2 se detallan las operaciones matemáticas disponibles para resolver las ecuaciones de Vladislavleva, especificando el nombre del conjunto de funciones y terminales (*Function Set*) correspondiente a cada caso.

En la Tabla 4.3, se especifican las ecuaciones, la cantidad de variables involucradas, y el conjunto de funciones disponibles correspondiente (*Function Set*). En la Figura 4.2 se pueden ver las curvas de nivel asociadas a las ecuaciones.

Function Set	Operaciones
Vladislavleva-A	$+, -, \times, \div, n^2$
Vladislavleva-B	$+, -, \times, \div, n^2, e^n, e^{-n}$
Vladislavleva-C	$+, -, \times, \div, n^2, e^n, e^{-n}$, sin, cos

Tabla 4.2: Operaciones en los conjuntos de funciones Vladislavleva

Nombre	Variables	Función	*Function Set*
Vladislavleva-1	2	$\frac{e^{-(x-1)^2}}{1{,}2+(y-2{,}5)^2}$	Vladislavleva-B
Vladislavleva-2	1	$e^{-x}x^3(\cos x \sin x)(\cos x \sin^2 x - 1)$	Vladislavleva-C
Vladislavleva-3	2	$e^{-x}x^3(\cos x \sin x)(\cos x \sin^2 x - 1)(y - 5)$	Vladislavleva-C
Vladislavleva-4	5	$\frac{10}{5+(x-3)^2+(y-3)^2+(z-3)^2+(v-3)^2+(w-3)^2}$	Vladislavleva-A
Vladislavleva-5	3	$\frac{30(x-1)(z-1)}{y^2(x-10)}$	Vladislavleva-A
Vladislavleva-6	2	$6\sin(x)\cos(y)$	Vladislavleva-B
Vladislavleva-7	2	$(x-3)(y-3)+2\sin((x-4)(y-4))$	Vladislavleva-C
Vladislavleva-8	2	$\frac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	Vladislavleva-A

Tabla 4.3: Funciones de Vladislavleva y la cantidad de variables de cada una. Ver la Tabla 4.2 para los Function Set de cada una. Ver la Tabla 4.4 para las características de los conjuntos generados.

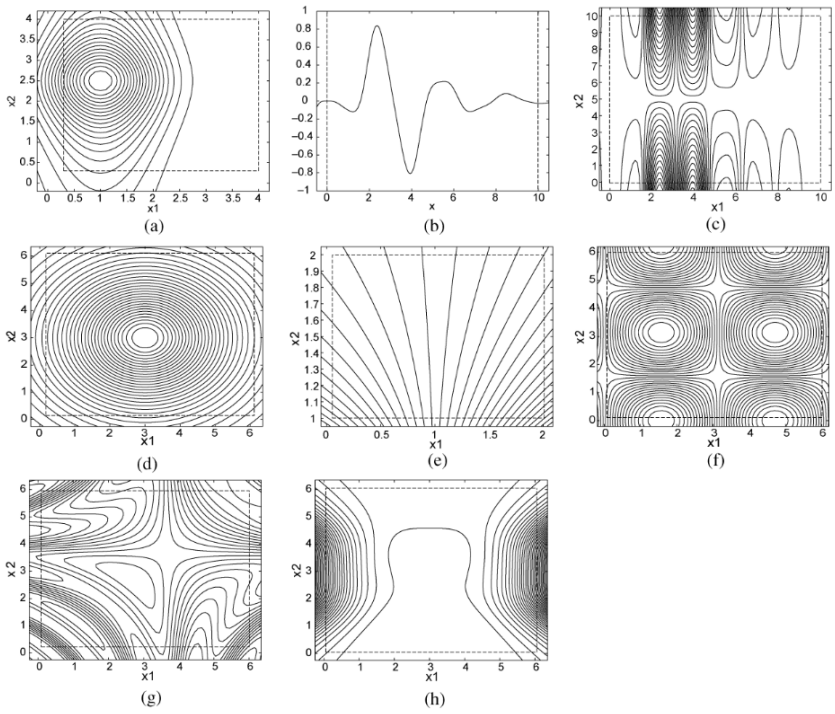

Figura 4.2: Curvas de nivel de las funciones objetivo (o sus proyecciones) del *benchmark* Vladislavleva en las regiones de prueba. Los gráficos siguen el orden correspondiente a las funciones presentes en la Tabla 4.3. Las líneas discontinuas representan los límites de las regiones de entrenamiento. Se puede observar que ninguna de las superficies de respuesta objetivo muestra un comportamiento patológico en la región de validación. En (d) $x_3 = x_4 = x_5 = 0$. En (e) $x_3 = 2$. Imagen extraída de Vladislavleva y cols. (2009).

Finalmente, en la Tabla 4.4, se describen las características de los conjuntos de entrenamiento y validación generados de manera independiente. En dicha tabla se puede notar que el conjunto de validación extiende en todos los casos la región cubierta por el conjunto de entrenamiento. A su vez, cabe recordar que $U[a, b, c]$ son c muestras aleatorias uniformes extraídas de a a b, ambas inclusive, para la variable. Mientras que $E[a, b, c]$ es una recta de puntos espaciados uniformemente, para cierta variable, con un intervalo de c, de a a b inclusive. Esta configuración permite generar *datasets* cuyos puntos forman una cuadrícula equiespaciada.

Nombre	Espacio de entradas para entrenamiento	Espacio de entradas para validación
Vladislavleva-1	U[0.3, 4, 100]	E[-0.2, 4.2, 0.1]
Vladislavleva-2	E[0.05, 10, 0.1]	E[-0.5, 10.5, 0.05]
Vladislavleva-3	x: E[0.05, 10, 0.1] y: E[0.05, 10.05, 2]	x: E[-0.5, 10.5, 0.05] y: E[-0.5, 10.5, 0.5]
Vladislavleva-4	U[0.05, 6.05, 1024]	U[-0.25, 6.35, 5000]
Vladislavleva-5	x: U[0.05, 2, 300] y: U[1, 2, 300] z: U[0.05, 2, 300]	x: E[-0.05, 2.1, 0.15] y: E[0.95, 2.05, 0.1] z: E[-0.05, 2.1, 0.15]
Vladislavleva-6	U[0.1, 5.9, 30]	E[-0.05, 6.05, 0.02]
Vladislavleva-7	U[0.05, 6.05, 300]	U[-0.25, 6.35, 1000]
Vladislavleva-8	U[0.05, 6.05, 50]	E[-0.25, 6.35, 0.2]

Tabla 4.4: Funciones de Vladislavleva y las características de sus conjuntos generados. $U[a, b, c]$ son c muestras aleatorias uniformes extraídas de a a b, ambas inclusive, para una variable. $E[a, b, c]$ es una recta de puntos espaciados uniformemente (para esta variable) con un intervalo de c, de a a b inclusive. Los conjuntos de validación y de entrenamiento son independientes.

4.4.4. Funciones de Ingeniería de procesos químicos

En las pruebas experimentales llevadas a cabo en un trabajo relacionado (J. Ferreira y cols., 2023), se incluyeron cuatro casos de estudios de sistemas de procesos químicos del mundo real (J. Ferreira, 2022). Dichos esquemas son presentados en la Figura 4.3. Estos pueden ser resueltos analíticamente y, por lo tanto, se puede obtener la expresión exacta de las funciones de salida, como se especifica en Seborg, Edgar, Mellichamp, y Doyle (2016).

La Figura 4.4 muestra el diagrama de flujo de los cuatro sistemas propuestos. Los semicírculos representan tanques de reactor cuyas flechas incidentes y salientes son las entradas y salidas del proceso químico, respectivamente. Los círculos representan puntos de mezcla en los que las entradas se mezclan produciendo una nueva salida, mientras que $A \underset{k1}{\longrightarrow} B$ indica que el reactivo A se convierte en el producto B a una tasa de reacción con constante cinética $k1$.

Para el Esquema 1, se consideran un modelo dinámico de un reactor continuo de tanque agitado (CSTR según su sigla en inglés) con tres reacciones de primer orden en serie ($A \rightarrow B \rightarrow C \rightarrow D$). Para el Esquema 2, se utilizan tres CSTRs en estado estacionario con reacción de primer orden ($A \rightarrow B$) en paralelo con un

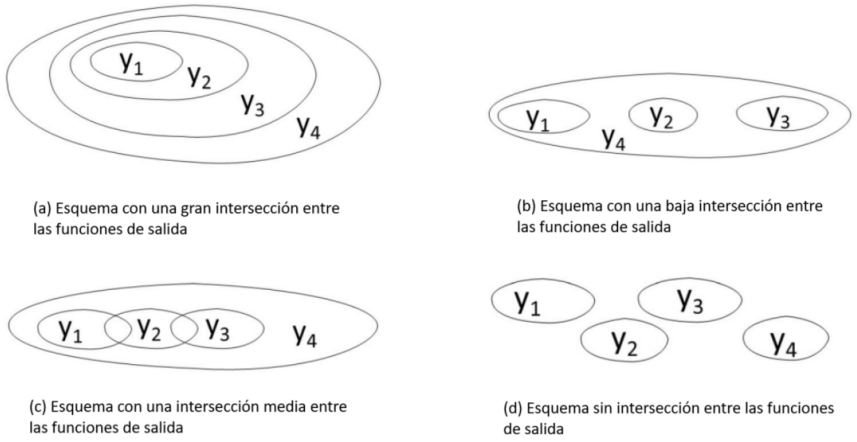

Figura 4.3: Diagramas de Venn que muestran los diferentes esquemas de compartición de términos. Todos los esquemas tienen cuatro funciones de salida y_1, y_2, y_3, e y_4. Imagen extraída de J. Ferreira y cols. (2023).

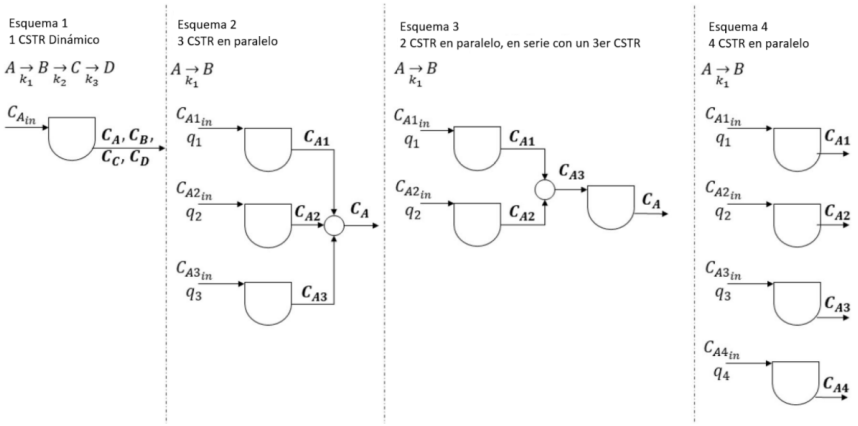

Figura 4.4: Casos prácticos de procesos químicos según los esquemas de compartición de términos presentados en la Figura 4.3. Imagen extraída de J. Ferreira y cols. (2023).

punto de mezcla para combinar las salidas. El Esquema 3 considera dos CSTRs en estado estacionario en paralelo y un tercero en serie con los otros dos (los tres reactores tienen reacciones de primer orden ($A \rightarrow B$)); y finalmente, el Esquema 4 considera cuatro CSTRs en estado estacionario en paralelo. Las Tablas 4.5,

4.6, 4.7 y 4.8 presentan la expresión analítica de las funciones de salida de cada caso de proceso químico (CP, *Chemical Process*) mostrado en la Figura 4.4.

CP (*Chemical Process*)	**Función de salida**
CP1	$\frac{dCA}{dt} = \frac{q}{V}(CA_{in} - CA) - k_1 CA$
CP2	$\frac{dCB}{dt} = \frac{q}{V}(CB + k_1 CA - k_2 CB)$
CP3	$\frac{dCC}{dt} = \frac{q}{V}(CC + k_2 CB - k_3 CC)$
CP4	$\frac{dCD}{dt} = \frac{q}{V}(CD + k_3 CC)$

Tabla 4.5: Funciones de salida para el sistema de procesos químicos con intercambio de términos del Esquema 1

CP (*Chemical Process*)	**Función de salida**
CP1	$CA_1 = CA_{1,in} \cdot \frac{q_1}{q_1 + k_1 V}$
CP2	$CA_2 = CA_{2,in} \cdot \frac{q_2}{q_2 + k_1 V}$
CP3	$CA_3 = CA_{3,in} \cdot \frac{q_3}{q_3 + k_1 V}$
CP4	$CA = \frac{CA_1 q_1 + CA_2 q_2 + CA_3 q_3}{q_1 + q_2 + q_3}$

Tabla 4.6: Funciones de salida para el sistema de procesos químicos con intercambio de términos del Esquema 2

CP (*Chemical Process*)	**Función de salida**
CP1	$CA_1 = CA_{1,in} \cdot \frac{q_1}{q_1 + k_1 V}$
CP2	$CA_2 = CA_{2,in} \cdot \frac{q_2}{q_2 + k_1 V}$
CP3	$CAs = \frac{CA_1 \cdot \frac{q_1}{q_1 + q_2} + CA_2 \cdot \frac{q_2}{q_1 + q_2}}{q_1 + q_2}$
CP4	$CA = \frac{CAs \cdot (q_1 + q_2)}{q_1 + q_2 + k_1 V}$

Tabla 4.7: Funciones de salida para el sistema de procesos químicos con intercambio de términos del Esquema 3

CP (*Chemical Process*)	**Función de salida**
CP1	$CA_1 = CA_{1,in} \cdot \frac{q_1}{q_1 + k_1 V}$
CP2	$CA_2 = CA_{2,in} \cdot \frac{q_2}{q_2 + k_1 V}$
CP3	$CA_3 = CA_{3,in} \cdot \frac{q_3}{q_3 + k_1 V}$
CP4	$CA_4 = CA_{4,in} \cdot \frac{q_4}{q_4 + k_1 V}$

Tabla 4.8: Funciones de salida para el sistema de procesos químicos con intercambio de términos del Esquema 4

En J. Ferreira y cols. (2023), se destaca que estos casos de estudio de procesos químicos son representativos de problemas significativos que actualmente

se están abordando utilizando modelos híbridos, que integran modelos mecanicistas con técnicas de aprendizaje automático. Por ejemplo, Chakraborty, Sivaram, Samavedham, y Venkatasubramanian (2020) y Chakraborty, Sivaram, y Venkatasubramanian (2021) abordan varios escenarios reales de este tipo de problemas con una complejidad similar a la de los escenarios utilizados en el anterior trabajo referenciado.

Los conjuntos de datos de entrada-salida se obtienen considerando las soluciones analíticas de cada variable de salida para cada caso de estudio. Se utilizaron los siguientes valores: $V = 1$ L, $k1 = 3{,}7$ min^{-1}, $k2 = 4$ min^{-1}, $k3 = 3$ min^{-1}, $q = 2$ L.min^{-1}, $CA(0) = CB(0) = CC(0) = CD(0) = 0$ mol.L^{-1}, $CA1_{in} = 1$ mol.L^{-1}, $CA2_{in} = 2$ mol.L^{-1}, $CA3_{in} = 3$ mol.L^{-1}, y $CA4_{in} = 4$ mol.L^{-1}.

En el trabajo citado, se generaron 100 puntos de manera aleatoria utilizando una distribución uniforme con los siguientes rangos: $CA_{in} \in [0{,}5\text{–}3]$ mol.L^{-1}, $t_2 \in [0\text{–}3]$ min, $q_i \in [0{,}5\text{–}4]$ L.min^{-1}. Al igual que en los conjuntos de funciones anteriores, los conjuntos de entrenamiento y validación son independientes.

Para obtener las expresiones analíticas del Esquema 1, se incorporan los valores de las variables utilizadas con el objetivo de simplificar los cálculos. Posteriormente, se presenta el desarrollo de las ecuaciones diferenciales de dicho esquema, aplicando seguidamente la fórmula para la resolución de ecuaciones diferenciales ordinarias lineales de primer orden.

Bajo la hipótesis de que $x(t)$ verifica la ecuación diferencial $\dot{x} + a(t)x = b(t)$, la solución general de la ecuación diferencial es:

$$x(t) = e^{-\int a(t)\,dt}\left(\int e^{\int a(t)\,dt} b(t)\,dt + K\right) \tag{4.1}$$

donde K es una constante de integración.

A continuación, se aplica la Fórmula 4.1 al caso de C_A en la ecuación presentada en la Tabla 4.5, obteniendo así la expresión analítica de la función correspondiente.

$$\frac{dC_A}{dt} = \frac{q}{V}(C_{\text{Ain}} - C_A) - k_1 C_A = \frac{2}{1}(C_{\text{Ain}} - C_A) - 3{,}7C_A$$

Luego, se simplifica para llevar la ecuación diferencial a la forma normal

$$\frac{dC_A}{dt} + 5{,}7C_A = 2C_{\text{Ain}}$$

Aplicando la Fórmula 4.1 y resolviendo la integral, se obtiene que:

$$C_A = \frac{2}{5{,}7}C_{\text{Ain}} + Ke^{-5{,}7t}$$

Finalmente, teniendo en cuenta la siguiente condición inicial:

$$C_A(0) = 0 = \frac{2}{5{,}7}C_{\text{Ain}} + Ke^{-5{,}7t}$$

$$K = -\frac{2}{5{,}7} C_{\text{Ain}}$$

Y por lo tanto:

$$C_A = \frac{2}{5{,}7} C_{\text{Ain}} - \frac{2}{5{,}7} C_{\text{Ain}} e^{-5{,}7t}$$

Este procedimiento se realiza de la misma forma con las cuatro ecuaciones, obteniendo las expresiones indicadas en la Tabla 4.9.

Componente	**Función de Concentración**
$C_A(C_{A_{in}}, t)$	$\frac{2C_{A_{in}}}{5{,}7}\left(1 - e^{-5{,}7t}\right)$
$C_B(C_{A_{in}}, t)$	$C_{A_{in}}\left(4{,}1111e^{-6t} + 0{,}2164 - 4{,}3275e^{-5{,}7t}\right)$
$C_C(C_{A_{in}}, t)$	$C_{A_{in}}\left(-8{,}4571e^{-5t} - 16{,}4444e^{-6t} + 24{,}7285e^{-5{,}7t} + 0{,}1731\right)$
$C_D(C_{A_{in}}, t)$	$C_{A_{in}}\left(-1{,}0000e^{-2t} + 8{,}4571e^{-5t} + 12{,}3333e^{-6t} - 20{,}0501e^{-5{,}7t} + 0{,}2597\right)$

Tabla 4.9: Resolución analítica del sistema de ecuaciones diferenciales presentado en la Tabla 4.5

Capítulo 5

Análisis experimental siguiendo un enfoque monoobjetivo

En este capítulo se presenta la evaluación experimental considerando solamente la minimización del error como objetivo, realizada para los algoritmos DSO, Operon y DSR sobre los *benchmarks* Vladislavleva, Feynman y las ecuaciones de IPQ, anteriormente presentados.

5.1. Configuración paramétrica

En la Tabla 5.1 se presentan los valores de hiperparámetros utilizados en todas las ejecuciones. La mayoría de los valores fueron seleccionados tomando como referencia estudios previos, aunque algunos fueron ajustados para garantizar una comparación justa entre los algoritmos evaluados. Un ejemplo de esto es la diferencia en el número de muestras entre DSO y DSR, donde se ajustó el número máximo de iteraciones a 500 en ambos casos, asegurando así que, en ausencia de una expresión que alcanzara el valor máximo de *reward*, los algoritmos tuvieran un límite comparable para detenerse.

En cuanto a la mencionada función de *reward*, tanto DSO como DSR utilizan *Inverse RMSE*, explicada anteriormente en la Sección 2.4.6. Por otro lado, Operon utiliza RMSE como función de *fitness*.

Por otro lado, como DSO utiliza 500 iteraciones, y en cada iteración ejecuta 10 generaciones de programación genética se establece como criterio de parada 5000 generaciones en el caso de Operon.

Parámetro	DSO	Operon	DSR
Parámetros Compartidos			
Tamaño del lote/población	100	100	100
Longitud mínima de expresión	4	4	4
Longitud máxima de expresión	64	65	64
Máximas expresiones	1.000.000	500.000	500.000
Máximas constantes	3	-	3
Función de recompensa/*fitness*	Inverse NRMSE	RMSE	Inverse NRMSE
Criterios de detención	500 iteraciones Recompensa ≈ 1	5000 generaciones Recompensa ≈ 1	500 iteraciones Recompensa ≈ 1
Número de ejecuciones paralelizadas	5	1	1
Parámetros de PG			
Generaciones	10	5000	-
Probabilidad de cruzamiento	0.5	1.0	-
Probabilidad de mutación	0.5	0.25	-
Tamaño del torneo	5	5	-
Método de población inicial	RNN	Balanced Tree Creator (BTC)	-
Generaciones por iteración/Iteraciones locales	25	0	-
Método de cruzamiento	One Point	Subtree	-
Método de mutación	Multiple	Single-point	-
Método de selección	Tournament	Tournament	-
Tamaño del torneo	5	5	-
Optimizador	L-BFGS-B (para const)	LM	-
Parámetros de RNN			
Optimizador	Adam	-	Adam
Tipo de neuronas	LSTM	-	LSTM
Número de capas	1	-	1
Neuronas por capa	32	-	32
Método de entrenamiento	PQT	-	PQT
Tamaño de la cola PQT	10	-	10
Tamaño de selección de muestra	1	-	1
Tasa de aprendizaje	0.0005	-	0.0005
Peso de la entropía	0.03	-	0.003

Tabla 5.1: Valores de los hiperparámetros

5.2. Procedimiento de los experimentos

5.2.1. Metodología para evaluar los algoritmos monoobjetivo

El procedimiento estadístico seguido se basa en la metodología aplicada en otros estudios relevantes, de áreas o temáticas similares (Derrac, Garcıa, Molina, y Herrera, 2011; Pedemonte, Luna, y Alba, 2018; Sheskin, 2011).

En primer lugar, se realizan 30 ejecuciones independientes para cada uno de los algoritmos. Para los tres *frameworks*, se utilizan los mismos conjuntos de datos de entrenamiento y validación para cada ejecución, que fueron generados inicialmente por el algoritmo DSO y exportados en formato `.csv` para ser empleados tanto por DSR como por Operon.

Para realizar la evaluación de los algoritmos, se seleccionaron métricas que ofrecen una visión integral de su desempeño. Específicamente, se reporta la mediana (Med), el mínimo (Min), el máximo (Max), y el rango intercuartílico (IQR) obtenido a partir de las 30 ejecuciones independientes. En cada ejecución, los algoritmos se entrenan con los datos de entrenamiento y luego se reportan las métricas obtenidas sobre los datos de validación.

La mediana se utiliza como una medida central que ofrece un valor representativo de los resultados, ya que es menos afectada por valores atípicos. Esto facilita una evaluación más sólida del rendimiento general del algoritmo a lo largo de las diferentes ejecuciones. El IQR se utiliza para medir la dispersión de los resultados, ofreciendo información sobre la variabilidad y consistencia del rendimiento. Un IQR reducido sugiere que el algoritmo produce resultados más confiables, lo que es deseado en contextos donde la estabilidad es de gran prioridad, como sucede en la industria de procesos químicos. Además, los valores mínimo y máximo aportan una perspectiva sobre el rango completo del desempeño de los algoritmos. La tasa de éxito es un indicador clave, ya que muestra si el algoritmo, en alguna de las ejecuciones independientes, logra descubrir la ecuación subyacente de manera satisfactoria. Por último, el tiempo promedio de ejecución (t_{avg}) es crucial para evaluar la eficiencia computacional de cada algoritmo.

En el capítulo anterior, se adelantó que, durante la evaluación del *benchmark* de Vladislavleva, se identificaron casos con valores de RMSE superiores a 100, infinitos o indefinidos debido a puntos problemáticos en el conjunto de validación. Para abordar estos casos, se recurrió al *Hall of Fame* de las ejecuciones, seleccionando iterativamente la siguiente expresión con el mejor RMSE en el conjunto de entrenamiento. Este proceso continuó hasta encontrar una expresión con un RMSE menor a 100 en la validación.

Sin embargo, se mantuvo un registro detallado de los casos con RMSE infinito, indefinido o mayor a 100 para realizar comparaciones posteriores. Aunque estos casos se documentaron para su análisis, fueron excluidos de los reportes finales, ya que su inclusión habría dificultado la interpretación clara de los diagramas, como los *boxplots* relacionados con el error.

No obstante, el anterior post-procesamiento de los resultados no fue necesario para los otros dos *benchmarks*, por lo que para cada ejecución independiente, se consideró la expresión matemática que logró un menor RMSE para los datos de entrenamiento.

5.2.2. Metodología para comparar los algoritmos mono-objetivo

Para comparar los resultados, se aplica el Test de Friedman como método general (*ómnibus*) para evaluar la distribución del RMSE en los puntos de validación. El Test de rangos de Friedman es un test estadístico no paramétrico utilizado para comparar las medianas de tres o más grupos relacionados, mediante la clasificación de datos en rangos y la evaluación de posibles diferencias

entre las muestras. Debido a razones de espacio, una explicación más detallada de este test se presentará en la Sección A.2 del anexo.

En caso de que el Test de Friedman detecte diferencias estadísticamente significativas, se utiliza el procedimiento *post-hoc* de Holm. Este método secuencial ajusta los p-valores de manera progresiva, comenzando por el más pequeño, y permite un mayor poder estadístico en comparación con la corrección de Bonferroni, sin aumentar el riesgo de errores de tipo I. Por razones de espacio, más detalles de este procedimiento se presentarán en la Sección del Anexo A.3. Es importante destacar que, si el *post-hoc* de Holm identifica diferencias significativas, se procede a comparar las medianas de RMSE de los algoritmos. En este contexto, un algoritmo se considerará superior a otro si presenta una mediana de RMSE en el conjunto de validación más baja. Debido a razones de espacio, una explicación más detallada de este test se presentará en la Sección del Anexo A.4.

Todas las pruebas estadísticas se realizan con un nivel de significancia del 95 %.

5.3. Resultados experimentales

Debido a la extensión de los resultados, estos se presentan en la Sección del Anexo A.5. En dicha sección, se incluye una tabla que recopila los resultados obtenidos para las 124 funciones evaluadas: 8 pertenecen al *benchmark* de Vladislavleva, 112 al de Feynman y 16 al conjunto de problemas de IPQ. Además, se presenta una tabla con los resultados de los *tests* de Friedman.

Por otro lado, en las Secciones del Anexo A.6, A.7 y A.8, se presentan los *boxplots* que muestran la distribución del RMSE en los puntos de validación para las ecuaciones evaluadas en cada conjunto de problemas.

En las Tablas 5.2, 5.3 y 5.4, se presenta un resumen de los casos en los que el RMSE alcanzó valores indefinidos o excesivamente grandes en 30 ejecuciones independientes para cada uno de los algoritmos (DSO, Operon y DSR respectivamente) en las ecuaciones del *benchmark* de Vladislavleva. El símbolo ∞ indica que el RMSE fue infinito, mientras que el símbolo >100 señala que el RMSE superó el valor de 100.

5.4. Análisis de los resultados

En el análisis de los resultados obtenidos de la evaluación de los algoritmos sobre diferentes conjuntos de datos, se observó que los tres algoritmos lograron recuperar la función matemática original en varios casos de los *benchmarks* Feynman y los problemas de IPQ. Sin embargo, para el *benchmark* de Vladislavleva, ninguno de los algoritmos fue capaz de obtener buenos resultados, como se muestra resumidamente en la Figura 5.1.

En el conjunto de datos de Feynman, se alcanzaron 38, 49 y 37 éxitos sobre 100 problemas posibles, para los algoritmos DSO, Operon y DSR, respectiva-

Ecuación	**RMSE=∞**	**RMSE>100**
1	5 [1,45,45,3,3]	5
2	0	0
3	0	0
4	0	0
5	1 [1]	0
6	0	0
7	2 [1,32]	0
8	2 [1,1]	2

Tabla 5.2: Resumen de casos en los que RMSE es infinito o indefinido, y en los que RMSE es mayor que 100 en 30 ejecuciones independientes para DSO en las ecuaciones de Vladislavleva. Para las soluciones que reportaron valores de RMSE infinito en el conjunto de validación, se indica cuántos puntos de tal conjunto no están definidos.

Ecuación	**RMSE=∞**	**RMSE>100**
1	9 [45,89,45,45,45,89,45,3,12]	3
2	0	0
3	3 [221,23,221]	0
4	0	2
5	0	0
6	0	7
7	2 [8,5]	0
8	0	0

Tabla 5.3: Resumen de casos en los que RMSE es infinito o indefinido, y en los que RMSE es mayor que 100 en 30 ejecuciones independientes para Operon en las ecuaciones de Vladislavleva. Para las soluciones que reportaron valores de RMSE infinito en el conjunto de validación, se indica cuántos puntos de tal conjunto no están definidos.

mente. De manera similar, en el conjunto de problemas de IPQ, se lograron 7, 7 y 6 éxitos sobre 16 problemas posibles, para los mismos algoritmos.

En cuanto a la evidente dificultad para resolver los problemas de RS en el *benchmark* de Vladislavleva, los resultados obtenidos eran esperables debido a la complejidad inherente al proceso de extrapolación, que consiste en predecir valores fuera del rango de datos observado durante el entrenamiento. En este *benchmark*, el conjunto de validación incluye puntos que se extienden más allá del rango cubierto por el conjunto de entrenamiento, lo que introduce un desafío significativo al requerir que los modelos generalicen más allá de las relaciones presentes en los datos de entrenamiento. Este comportamiento contrasta con los otros dos *benchmarks* evaluados, donde, aunque los conjuntos de entrenamiento y validación son independientes, los puntos de validación permanecen dentro de los mismos rangos que los del entrenamiento, evitando así el desafío asociado a

Ecuación	RMSE=∞	RMSE>100
1	3 [1,1,45]	0
2	0	0
3	1 [4]	0
4	0	0
5	0	0
6	0	2
7	0	0
8	1 [1]	4

Tabla 5.4: Resumen de casos en los que RMSE es infinito o indefinido, y en los que RMSE es mayor que 100 en 30 ejecuciones independientes para DSR en las ecuaciones de Vladislavleva. Para las soluciones que reportaron valores de RMSE infinito en el conjunto de validación, se indica cuántos puntos de tal conjunto no están definidos.

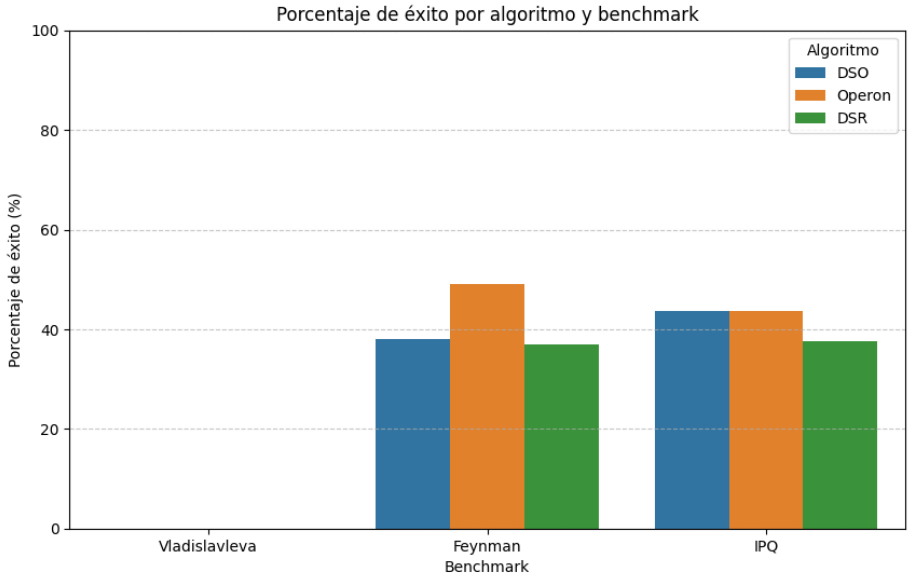

Figura 5.1: Número de éxitos por algoritmo.

la extrapolación.

Por esta razón, aunque en el conjunto de entrenamiento se lograron obtener valores de *fitness* comparables a los obtenidos en los otros *benchmarks*, al evaluar el error sobre el conjunto de validación, quedó en evidencia la dificultad de generalizar a puntos no vistos previamente, que podrían considerarse críticos o importantes al valer cero para algunas de las variables, y que están fuera del rango de los puntos utilizados para el entrenamiento.

Continuando el análisis de los resultados de los algoritmos sobre el *benchmark* de Vladislavleva, se genera una tabla similar a la presentada en Vladislavleva y cols. (2009), donde se reportaba el comportamiento patológico de ciertas ecuaciones en términos de RMSE. En dicho estudio, la segunda y tercera columna mostraban el porcentaje de ecuaciones que producían errores indefinidos o excesivamente grandes. Siguiendo este criterio, se consideró una ejecución como patológica cuando el RMSE era infinito o mayor a 100, pero en lugar de porcentajes, se reportó el número de ejecuciones (de un total de 30) en las que se presentaron estos comportamientos para la mejor solución en lo que respecta a minimizar el error en entrenamiento de la respectiva ejecución.

Como se observa en las Tablas 5.2, 5.3 y 5.4, los algoritmos exhibieron comportamientos problemáticos en diversas ejecuciones para ciertas ecuaciones específicas. En particular, Operon registró el mayor número de ejecuciones con RMSE indefinido o infinito en el conjunto de validación, alcanzando un total de 14 casos, seguido de DSO con 10 casos, mientras que DSR presentó este problema en menor medida, con un total de 5 casos. La única ecuación que resultó problemática para los tres algoritmos fue la primera ecuación, mientras que las ecuaciones 2, 4 y 6 no registraron puntos indefinidos en ningún caso.

En relación con los valores de RMSE superiores a 100, Operon nuevamente lideró con 12 ejecuciones, seguido de DSO con 7 casos y DSR con 6 casos. No se identificó ninguna ecuación en la que los tres algoritmos enfrentaran simultáneamente esta problemática, y en la mitad de las ecuaciones no hubo registros de RMSE altos para ninguno de ellos. Además, solo la segunda ecuación no presentó ni valores indefinidos ni RMSE elevados en las ejecuciones de los algoritmos evaluados.

Estos resultados destacan una mayor tendencia de Operon y DSO a generar predicciones problemáticas en comparación con DSR, aunque ninguno de los algoritmos estuvo exento de problemas al extrapolar.

Es importante destacar que se realizaron los mismos análisis para los otros dos *benchmarks*, sin encontrar casos en los que el RMSE en el conjunto de validación superara el valor de 100, ni situaciones donde la función obtenida resultara indefinida. Este comportamiento puede atribuirse principalmente a que los conjuntos de entrenamiento y validación cubrían el mismo dominio, lo que favoreció una mayor estabilidad en los resultados, como se mencionó anteriormente.

Es importante señalar que, para la Ecuación 6 de Vladislavleva, que corresponde a $f(x, y) = 6 \times sin(x)cos(y)$, se presentó una limitación importante: en el trabajo original de Vladislavleva y cols. (2009), las funciones seno y coseno no estaban incluidas en el conjunto de operadores disponibles para la RS. Como resultado, era imposible que los algoritmos recuperaran exactamente la ecuación original bajo esas condiciones. Sin embargo, al incorporar estas funciones en el conjunto de operadores disponibles para los tres algoritmos, se observa que todos ellos fueron capaces de resolver el problema con éxito, logrando recuperar la ecuación original sin inconvenientes.

El resultado anterior resalta la importancia de contar con conjuntos de funciones y terminales bien diseñados en RS. Si bien la ampliación de estos conjun-

tos puede resultar beneficiosa, es crucial realizar un análisis cuidadoso sobre qué funciones y constantes son realmente relevantes para el problema en cuestión. Incluir funciones o terminales inapropiados puede desviar la exploración del algoritmo hacia subespacios de soluciones innecesarios, lo que dificulta el hallazgo de la solución exacta. Además, la ausencia de funciones clave, como ciertas constantes comúnmente presentes en ecuaciones de la ingeniería de procesos, puede limitar la capacidad del algoritmo para generar soluciones precisas, forzándolo a obtener solo aproximaciones o a intentar construir soluciones complejas, como polinomios de Taylor equivalentes. Esta última opción resulta poco viable si el número de iteraciones disponibles es limitado.

En línea con lo mencionado en el estudio La Cava y cols. (2021), nuestros experimentos confirmaron que el algoritmo Operon logró obtener numerosos éxitos en la recuperación de la función original para el *dataset* de Feynman. Al igual que en dicho estudio, Operon mostró un rendimiento destacado en términos de precisión, alcanzando altos valores de R^2 y bajos valores de RMSE, reflejado en la Tabla A.3, en el caso de las ecuaciones de Feynman.

Sin embargo, a menudo fue necesario inspeccionar visualmente los resultados de las ejecuciones de Operon y considerar su versión simplificada mediante *SymPy* como se explicó en la Sección 4.1.3, para identificar con mayor claridad el éxito. En muchos casos, las expresiones generadas por el algoritmo involucraban múltiplos de π, tanto al multiplicar como al dividir.

Además, se observa que muchas de las expresiones incluían sumas de valores muy pequeños, cercanos a cero, que podían ser ignorados sin comprometer la precisión del modelo. En estos casos, la obtención de la expresión podía considerarse exitosa. Estos detalles refuerzan la idea de que, aunque los modelos de Operon tienden a ser más complejos, en numerosas ocasiones su forma final requería sólo ajustes menores para ser considerada simbólicamente equivalente a la función original.

Debido a los argumentos de los párrafos anteriores, aunque Operon produjo modelos altamente precisos, estos tienden a ser más complejos y de mayor tamaño en comparación con los generados por DSO y DSR, lo que coincide con lo reportado en La Cava y cols. (2021).

En cuanto a los tiempos de ejecución, Operon completaba cada ejecución en aproximadamente 10 segundos. En cambio, DSR tardaba alrededor de 230 segundos por ejecución. Por su parte, DSO finalizaba cinco ejecuciones en aproximadamente 1685 segundos, aprovechando la capacidad de ejecutarlas en paralelo. Sin embargo, este enfoque no resultaba ventajoso para DSR debido a la naturaleza de su algoritmo, que no se beneficiaba del paralelismo. De todas formas, si se optara por ejecutar una única instancia de DSO, el tiempo promedio de ejecución sería de aproximadamente 900 segundos.

En la Tabla A.4 se observa que no existen diferencias estadísticas significativas entre las distribuciones obtenidas por los distintos algoritmos en un solo problema de los *benchmarks* de Vladislavleva e IPQ (específicamente, Vladislavleva 1 e IPQ Esq1.CP2), así como en cuatro ecuaciones del conjunto de problemas del *benchmark* de Feynman (en particular, I.34.1, I.44.4, III.8.54 y III.21.20).

En las Secciones A.9, A.10 y A.11 del Anexo se presentan los resultados del procedimiento *post-hoc* de Holm para las ecuaciones donde la prueba de Friedman indicó que las distribuciones de valores de RMSE eran estadísticamente diferentes.

A partir de estas pruebas estadísticas, cuyos resultados se encuentran resumidos en la Tabla 5.5, se puede observar una clara tendencia en la que Operon ofrece los mejores resultados generales para los tres *benchmarks* evaluados, en comparación con los otros dos algoritmos.

Sin embargo, al comparar DSO y DSR, no se encuentra una tendencia tan clara. Para el *benchmark* de Vladislavleva, en los 7 casos no hubo diferencias estadísticas entre los algoritmos. En el *benchmark* de Feynman, hubo 70 casos sin diferencias estadísticas, con DSO superando a DSR en 4 ocasiones y siendo superado en 22. Finalmente, para los problemas de IPQ, se encontraron 12 casos sin diferencias estadísticas, con 2 ocasiones donde DSO superaba a DSR, mientras que este otro lo superaba en un solo caso.

Adicionalmente, se recuerda que al analizar la capacidad para recuperar la función matemática original, DSO superó a DSR en un caso tanto en el *benchmark* de Feynman como en el de IPQ. Esto sugiere que las diferencias podrían ser más evidentes en situaciones donde el algoritmo no logra un éxito completo.

Benchmark	**DSO vs Operon**		**Operon vs DSR**		**DSO vs DSR**	
Vladislavleva	0	6	6	0	0	0
Feynman	21	68	65	22	4	22
IPQ	0	15	15	0	2	1
Total (%)	**17.8 %**	**75.4 %**	**72.9 %**	**18.6 %**	**5 %**	**19.5 %**

Tabla 5.5: Resumen de resultados obtenidos al comparar los algoritmos DSO, Operon y DSR utilizando el procedimiento *post-hoc* de Holm para los distintos *benchmarks*.

Por otro lado, un aspecto relevante a destacar es que la reproducción de los resultados reportados en la bibliografía resultó ser más compleja de lo anticipado, incluso cuando se emplearon configuraciones de hiperparámetros similares o idénticas. Este desafío de reproducibilidad puede estar relacionado con varios factores, entre ellos las variaciones en los conjuntos de datos utilizados. A pesar de que los conjuntos de datos en los experimentos abarcaban rangos y cantidades de puntos prácticamente idénticos, las diferencias sutiles entre ellos, especialmente en los puntos seleccionados para el entrenamiento y la validación, pudieron haber influido significativamente en los resultados. Como se mencionó anteriormente, en algunos casos ciertos puntos de validación resultaron ser más problemáticos que otros. En el caso específico del *benchmark* de Vladislavleva, la cantidad de puntos en validación fuera del rango utilizado para el entrenamiento se genera de manera aleatoria, ya que los *datasets* son creados seleccionando puntos dentro de un rango determinado de forma aleatoria. Este factor introduce una variabilidad que puede haber afectado la replicabilidad de los resultados.

A pesar de estas dificultades, el rendimiento de DSO en los problemas de RS fue razonablemente bueno, lo que motivó la decisión de continuar experimentando con este algoritmo, particularmente en el contexto del enfoque multiobjetivo. La principal ventaja de DSO, además de sus buenos resultados, radica en su componente de aprendizaje profundo, el cual no está presente en Operon y le confiere a DSO un potencial adicional para abordar problemas complejos.

Capítulo 6

Análisis experimental para el enfoque multiobjetivo

En este capítulo se presenta la evaluación experimental realizada para el algoritmo multiobjetivo DSO-MO (ver Sección 4.3), con el fin de poder ejecutarlo teniendo en consideración *shape constraints* para un subconjunto de problemas del *benchmark* de Feynman, y la cantidad de puntos indefinidos para un subconjunto de problemas del *benchmark* de Vladislavleva, además del error.

6.1. Metodología para evaluar el algoritmo multiobjetivo

En la evaluación del algoritmo multiobjetivo, se siguió un enfoque sistemático similar al utilizado para evaluar los algoritmos monoobjetivo. Tras realizar las ejecuciones independientes, se calcula la mediana, el mínimo, el máximo y el IQR para los resultados obtenidos.

Adicionalmente, se reportan la cantidad de puntos indefinidos en el conjunto de validación para un subconjunto de ecuaciones de Vladislavleva. Para un análisis más exhaustivo, se considera si esta cantidad varía al eliminar del conjunto aquellos puntos donde alguna variable x_i es igual a cero.

Por otra parte, para evaluar el cumplimiento de restricciones de monotonicidad en datos de validación, se reportan estadísticas clave como la media, la mediana, el IQR y el mínimo del porcentaje de puntos que satisfacen la condición de monotonicidad para ciertas variables, en un subconjunto particular de ecuaciones de Feynman. Estas métricas permiten cuantificar el desempeño de los algoritmos en relación con las propiedades deseadas, facilitando así comparaciones entre los enfoques de DSO y DSO-MO en la sección de resultados.

6.2. Problemas seleccionados

6.2.1. Problemas para evaluar *shape constraints*

Siguiendo el criterio establecido en Kronberger y cols. (2021), se utiliza un subconjunto de instancias de problemas de la *Feynman Symbolic Regression Database*. Se selecciona un subconjunto de las instancias evaluadas en el trabajo mencionado, específicamente aquellas para las cuales es posible derivar restricciones de monotonicidad de forma sencilla y donde, al menos en un punto del conjunto de validación, alguna de estas restricciones no se cumplió en la ecuación generada por DSO con el menor RMSE en entrenamiento. Las funciones seleccionadas se indican en la Tabla 6.1. Estas funciones son suaves y no lineales.

Para cada instancia, se definieron varias *shape constraints* derivadas de las expresiones conocidas. Las *shape constraints* se indican en la Tabla 6.2, utilizando una notación compacta en forma de tupla. El primer elemento de la tupla es el rango permitido para la salida del modelo, y los elementos restantes indican, para cada variable de entrada, si la derivada parcial del modelo con respecto a esa variable es no positiva (-1, monótona decreciente) o no negativa (1, monótona creciente). El valor 0 indica que no hay restricciones sobre la derivada parcial.

Instancia	**Espacio de entrada**
I.6.2	$(\sigma, \theta) \in [1.,3]^2$
I.9.18	$(x_1, y_1, z_1, m_1, m_2, G, x_2, y_2, z_2) \in [3.,4]^3 \times [1.,2]^6$
I.32.17	$(\epsilon, c, E_f, r, \omega, \omega_0) \in [1.,2]^5 \times [3.,5]$
I.41.16	$(\omega, T, h, k_b, c) \in [1.,5]^5$
II.6.15a	$(\epsilon, pd, r, x, y, z) \in [1.,3]^6$
II.11.27	$(n, \alpha, \epsilon, E_f) \in [0.,1]^2 \times [1.,2]^2$
II.35.21	$(n_\rho, \mu_m, B, k_b, T) \in [1.,5]^5$
III.9.52	$(pd, E_f, t, h, \omega, \omega_0) \in [1.,3]^4 \times [1.,5]^2$
III.10.19	$(\mu_m, B_x, B_y, B_z) \in [1.,5]^4$

Tabla 6.1: Funciones del *dataset* Feynman utilizadas para el análisis multiobjetivo con *shape constraints*. La columna de espacio de entrada se refiere a los dominios de las variables.

6.2.2. Problemas al evaluar puntos indefinidos

Se realizó una nueva evaluación de las ecuaciones de Vladislavleva 1, 5, 7 y 8, que previamente mostraron la presencia de puntos indefinidos al emplear el algoritmo DSO. Para esta reevaluación, se utilizó una versión modificada de DSO con soporte multiobjetivo, y foco tanto en minimizar el *RMSE* como en reducir la cantidad de puntos indefinidos. Recordar que también se clasifican como puntos indefinidos aquellos cuya imagen toma valores negativos.

Instancia	Restricciones
I.6.2	$([0..\infty], 0, -1)$
I.9.18	$([0..\infty], -1, -1, -1, 1, 1, 1, 1, 1, 1)$
I.32.17	$([0..\infty], 1, 1, 1, 1, 1, -1)$
I.41.16	$([0..\infty], 0, 1, -1, 1, -1)$
II.6.15a	$([0..\infty], -1, 1, -1, 1, 1, 1)$
II.11.27	$([0..\infty], 1, 1, 1, 1)$
II.35.21	$([0..\infty], 1, 1, 1, -1, -1)$
III.9.52	$([0..\infty], 1, 1, 0, -1, 0, 0)$
III.10.19	$([0..\infty], 1, 1, 1, 1)$

Tabla 6.2: *Shape constraints* utilizadas en cada instancia de problema. La columna de restricciones consiste en tuplas donde el primer elemento es el rango permitido para la salida del modelo y los elementos restantes corresponden a las variables. Un valor de 1 representa una restricción no decreciente, -1 no creciente, y 0 cuando no hay restricción.

6.3. Configuración paramétrica

Para los siguientes experimentos ejecutados utilizando el algoritmo DSO-MO, se replicó el entorno de pruebas descrito en la Sección 4.1.1.

Por otra parte, la incorporación de las *shape constraints* como un nuevo objetivo en la optimización multiobjetivo provocó un aumento en los tiempos de ejecución. Para abordar este problema, se realizó un ajuste en los datos utilizados en las ejecuciones que incluían *shape constraints*.

En particular, se tuvo que reducir el tamaño de los *datasets* de entrenamiento y validación a 500 puntos, para el subconjunto seleccionado de problemas de Feynman. El resto de hiperparámetros mantuvieron sus valores respecto a los presentados en la Tabla 5.1 para DSO.

En cuanto a la evaluación de puntos indefinidos, los parámetros también se mantuvieron consistentes con los valores establecidos anteriormente en la Tabla 5.1 para DSO. Asimismo, para las ecuaciones de Vladislavleva, se reutilizaron los conjuntos de datos de entrenamiento y validación originales.

Es relevante mencionar nuevamente que para los anteriores problemas, el conjunto de validación extiende al de entrenamiento, incluyendo el vector nulo y otros puntos donde algunas coordenadas tienen valores de cero, situación que no se presenta en el conjunto de entrenamiento. Esta decisión fue intencional, dado que estos dominios han sido utilizados en trabajos previos, garantizando que los resultados obtenidos sean comparables con estudios anteriores.

6.4. Resultados experimentales

6.4.1. Evaluación considerando la minimización del RMSE y la cantidad de puntos indefinidos

Siguiendo el enfoque de los experimentos reportados en la Sección 5.3, en la Tabla 6.3 se utiliza el símbolo ✓ en la columna *éxito* para indicar cuando un algoritmo logró obtener, en al menos una de sus 30 ejecuciones independientes, una expresión matemática equivalente a la original. También se considera un éxito cuando el algoritmo logra un error para el conjunto de entrenamiento igual a cero o muy cercano a este valor, aunque dicha expresión no sea exactamente equivalente a la buscada. En caso contrario, se utiliza el símbolo × para señalar la falta de éxito.

Dado que ahora se consideran dos objetivos simultáneamente, la selección de la solución representativa de cada ejecución independiente se realiza después de un análisis de compromiso (*trade-off*) entre el cumplimiento de las restricciones y el error en los datos de entrenamiento. Este proceso se aplica únicamente cuando el frente de Pareto contiene más de una solución, permitiendo una evaluación más equilibrada y acorde a los objetivos planteados.

Función	**DSO**			**DSO MO**		
	Med	**IQR**	**Éxito**	**Med**	**IQR**	**Éxito**
Vladislavleva 1	1.29e-01	1.38e-02	×	**1.26e-01**	2.26e-02	×
Vladislavleva 5	6.33e-01	1.56e-01	×	**5.90e-01**	1.11e-01	×
Vladislavleva 7	4.30e+00	2.31e-01	×	**4.20e+00**	2.73e-01	×
Vladislavleva 8	**2.30e+00**	6.06e+00	×	2.46e+00	9.07e+00	×

Tabla 6.3: Mediana (Med), rango intercuartílico (IQR) y éxito en la tarea de descubrir la ecuación original para cada función y algoritmo.

Función	**DSO**			**DSO MO**		
	Min	**Max**	t_{avg}	**Min**	**Max**	t_{avg}
Vladislavleva 1	**7.47e-02**	3.27e-01	1.74e+03	1.06e-01	1.18e+02	2.39e+03
Vladislavleva 5	**3.85e-01**	3.93e+00	2.03e+03	5.06e-01	5.50e+00	2.50e+03
Vladislavleva 7	3.79e+00	1.01e+01	1.56e+03	**3.47e+00**	1.11e+02	1.69e+03
Vladislavleva 8	2.03e+00	7.11e+01	2.03e+03	**1.72e+00**	1.89e+02	2.18e+03

Tabla 6.4: Mínimo (Min), máximo (Max) de la distribución de RMSE y tiempo promedio en segundos (t_{avg}) para cada función y algoritmo.

En cuanto a las medianas del RMSE, DSO-MO no muestra una mejora significativa en comparación con DSO. Para las funciones evaluadas, la mediana del RMSE de DSO-MO es menor en tres casos respecto a las obtenidas con DSO. Sin embargo, DSO-MO presenta un IQR ligeramente mayor en tres funciones, lo que indica una mayor variabilidad en los resultados de las ejecuciones. Los valores mínimos y máximos de RMSE tampoco muestran una ventaja clara de DSO-MO. En dos ecuaciones, DSO-MO logró reducir el valor mínimo en una de las 30 ejecuciones, pero en las otras dos ecuaciones no se observó tal mejora.

En cuanto al éxito en la recuperación de expresiones originales, ningún algoritmo consiguió recuperar una expresión equivalente a la original en ninguna

Ecuación	DSO	DSO - MO
	RMSE=∞	**RMSE=∞**
1	5 [1,45,45,3,3]	11 [3,1,1,45,1,251,3,1,45,1,45]
5	1 [1]	0
7	2 [1,32]	2 [1,2]
8	2 [1,1]	3 [1,1,1]

Tabla 6.5: Resumen de casos en los que RMSE es infinito o indefinido en 30 ejecuciones independientes para cada uno de los algoritmos (DSO, DSO-MO) y para un subconjunto de las ecuaciones de Vladislavleva. Se indica cuántos puntos del *dataset* de validación provocan dicho comportamiento.

Ecuación	DSO		DSO - MO	
	RMSE=∞ con $x_i \neq 0$	**RMSE>100**	**RMSE=∞ con $x_i \neq 0$**	**RMSE>100**
1	0	5	2	2
5	1	0	0	0
7	2	0	2	0
8	2	4	3	0

Tabla 6.6: Resumen de casos en los que RMSE es infinito con $x_i \neq 0$, y RMSE mayor que 100 en 30 ejecuciones independientes para los algoritmos DSO y DSO-MO, en un subconjunto de las ecuaciones de Vladislavleva.

de las 30 ejecuciones para las ecuaciones evaluadas.

El análisis de la Tabla 6.5 muestra los casos en los que el RMSE es infinito (∞), mientras que el de la Tabla 6.6 presenta los casos en los que se obtiene un valor de RMSE infinito con alguna variable $x_i \neq 0$, o un valor mayor que 100 para el error, tanto para DSO como para DSO-MO, evaluados en 30 ejecuciones independientes para un subconjunto de ecuaciones de Vladislavleva.

La columna del RMSE=∞ con $x_i \neq 0$ es relevante debido a que el conjunto de validación expande al conjunto de entrenamiento. Esto significa que muchos de los puntos indefinidos reportados en validación se deben a que el modelo no fue capaz de entrenarse dentro del mismo rango del conjunto de entrenamiento, es decir, que los modelos generados no extrapolan bien. No obstante, la tabla también refleja que no todos los valores indefinidos se deben a que x_i sea cero, ya que algunos casos surgen incluso cuando $x_i \neq 0$.

En cuanto a los casos en los que el RMSE es infinito, DSO presenta un total de 5 casos en la Ecuación 1, mientras que DSO-MO muestra un aumento considerable, con 11 casos. En DSO, estos 5 casos están distribuidos de la siguiente manera: 1 punto en una ocasión, 45 puntos en dos ocasiones y 3 puntos en otras dos. Por otro lado, en DSO-MO se observan 11 ocurrencias, con una mayor variabilidad, distribuyéndose entre 1 y 251 puntos que provocan el RMSE infinito.

Para los casos en los que el RMSE es infinito y $x_i \neq 0$, DSO no presenta casos

en la Ecuación 1, mientras que DSO-MO reporta 2 casos. Esta diferencia indica que algunos valores indefinidos en DSO-MO no se deben exclusivamente a que alguna variable x_i sea cero, lo que evidencia que el comportamiento indefinido también puede surgir en otros escenarios. No obstante, para esta primer ecuación se tuvieron 5 ejecuciones en DSO que arrojaron valores de RMSE muy altos en validación, y 2 en el caso de DSO-MO. De esta forma, se podría considerar que 10 de las 30 ejecuciones arrojaron resultados malos en DSO, y 13 por el lado de DSO-MO.

En la Figura 6.1 se presenta la representación gráfica de la primera ecuación del *benchmark* Vladislavleva, la cual ilustra el comportamiento de la función. En la Figura 6.2, se observa una de las soluciones obtenidas por el algoritmo DSO-MO, donde se marcan los tres puntos indefinidos, que corresponden a los siguientes valores: (0, 0), (0.1, -0.1) y (0.2, -0.2). Como mencionamos, por lo general estos puntos indefinidos no estaban contemplados en el rango del conjunto de entrenamiento.

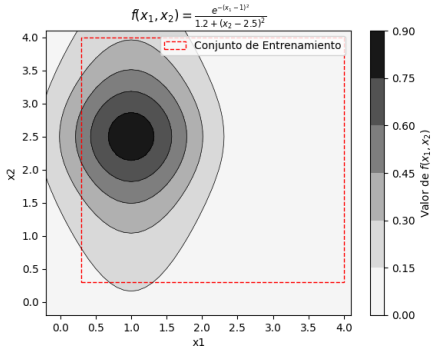

Figura 6.1: Representación gráfica de la primer ecuación del *benchmark* Vladislavleva.

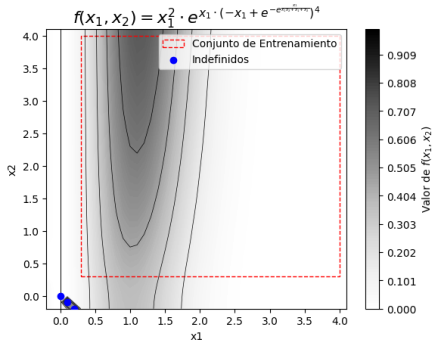

Figura 6.2: Solución obtenida por el algoritmo DSO-MO para la primer ecuación del *benchmark* Vladislavleva.

Por otro lado, en la Ecuación 5, se observa un comportamiento distinto. DSO muestra un solo caso en el que $x_i \neq 0$ produce un RMSE infinito, mientras que DSO-MO no reporta ningún caso con valores indefinidos.

En la ecuación 7, tanto DSO como DSO-MO muestran 2 casos en los que $x_i \neq 0$ provoca un RMSE infinito. Sin embargo, DSO presenta una mayor variabilidad con un caso en el que 32 puntos son responsables de dicho comportamiento, mientras que DSO-MO solo muestra 2 puntos indefinidos. Esto podría sugerir que DSO-MO ofrece una mayor estabilidad en esta ecuación.

En lo que respecta a los valores de RMSE mayores que 100, en la ecuación 8, DSO presenta 4 casos, mientras que DSO-MO no muestra ninguno. Este resultado favorece a DSO-MO, ya que parece controlar mejor los errores elevados en esta ecuación en particular. No obstante, para DSO-MO fueron 3 las ejecuciones independientes que reportaron puntos indefinidos pertenecientes al conjunto de validación, mientras que en DSO hubo un caso menos, todos ellos relativos a un

único punto.

Por otra parte, DSO-MO es más lento que DSO, como se refleja en los valores de t_{avg}, con tiempos promedio más altos en todas las funciones. Esto se debe a las modificaciones introducidas en DSO-MO, enfocadas en reducir los puntos indefinidos, implicando un costo computacional adicional al tener que manejar la dominancia de las distintas soluciones, debido al objetivo adicional que cuantifica la cantidad de puntos indefinidos pertenecientes al conjunto de entrenamiento.

6.4.2. Evaluación considerando la minimización del RMSE y las *shape constraints*

Al igual que en los experimentos reportados en la Sección 5.3, en la Tabla 6.7 se utiliza el símbolo ✓ en la columna *éxito* para señalar cuando un algoritmo logró obtener, en al menos una de sus 30 ejecuciones independientes, una expresión matemática equivalente a la original. También se considera exitoso si el algoritmo alcanza un error en el conjunto de entrenamiento igual a cero o muy cercano, aunque la expresión no sea exactamente equivalente a la buscada. En caso contrario, se emplea el símbolo × para denotar la falta de éxito. Además, la tabla reporta las medianas y los IQRs obtenidos.

Por otro lado, en la Tabla 6.8 se presentan los valores mínimos, máximos y el tiempo promedio de ejecución.

En cuanto a las medianas del RMSE, DSO-MO no muestra muchos casos de mejora en comparación con DSO en la minimización del error. Para las funciones evaluadas, la mediana del RMSE de DSO-MO es menor en tres de los nueve casos respecto a las obtenidas con DSO. A su vez, DSO-MO presenta un IQR mayor en ocho de las nueve funciones, lo que indica una mayor variabilidad en los resultados de las ejecuciones. Los valores mínimos y máximos de RMSE tampoco muestran una ventaja clara de DSO-MO. En cuatro de las nueve ecuaciones, DSO-MO logró reducir el valor mínimo en una de las 30 ejecuciones, pero en las otras cinco ecuaciones no se observó tal mejora. A su vez, el valor máximo aumenta en 8 casos para DSO-MO.

En lo que respecta al éxito en la recuperación de expresiones originales, ningún algoritmo logró recuperar una expresión equivalente a la original en ninguna de sus 30 ejecuciones para las ecuaciones evaluadas.

En relación con los frentes de Pareto obtenidos de las distintas ejecuciones independientes por ecuación, se observó que, en general, cada frente contenía una única solución: aquella con el menor RMSE que cumplía con las restricciones de monotonía correspondientes para el conjunto de entrenamiento. Este escenario era el preferido, siempre que los valores de RMSE obtenidos disminuyeran en comparación con los experimentos de la sección anterior. Sin embargo, como se puede observar en la Tabla 6.7, este comportamiento no se presentó en todos los casos.

Por otro lado, se identificaron situaciones en las que el frente de Pareto estaba compuesto por entre 2 y 4 soluciones no dominadas. En estos casos, el menor RMSE no se obtenía a partir de funciones monótonas, lo que llevó a

la decisión de analizar las otras soluciones que respetaban estas propiedades en mayor proporción. De este modo, se podían identificar en la mayoría de los casos, soluciones que satisfacían las restricciones, aunque con un leve incremento, no mayor del 5 %, en el error del conjunto de entrenamiento.

Este comportamiento es consistente con lo reportado en Kronberger y cols. (2021), donde al comparar la mediana del error entre los algoritmos NSGA-II y GSPC, se observó que NSGA-II, al aceptar soluciones no completamente factibles (aquellas que violan al menos una restricción), generaba soluciones ligeramente mejores en términos de error. Esto sugiere que permitir soluciones parcialmente no factibles puede, en ciertos casos, conducir a modelos con un desempeño superior en términos de error.

Como se puede observar en la Tabla 6.9, la introducción de este segundo objetivo permitió alcanzar una Mediana del 100 %, y un IQR de 0 % del porcentaje de puntos del conjunto de validación que cumplen la restricción de monotonía para todas las variables con restricciones de las ecuaciones consideradas.

Por otro lado, en el 71.4 % de las variables se mantuvo o mejoró la media del porcentaje de puntos en validación que cumplen con las restricciones. En cuanto al valor mínimo, este comportamiento se observó en el 54.5 % de las variables.

Para el problema I.9.18 del *benchmark* de Feynman, se comparan los resultados de los dos algoritmos mediante *boxplots* que ilustran el porcentaje de puntos de validación que cumplen con las restricciones de monotonicidad en las 9 variables de la función. En la Figura 6.3 se presentan los resultados de las 30 ejecuciones de DSO, y en la Figura 6.4 los correspondientes a las 30 ejecuciones de DSO-MO.

En el caso de DSO-MO, las mejoras significativas en el cumplimiento de las restricciones de monotonicidad se reflejan en *boxplots* que prácticamente se visualizan como líneas. Esto se debe a que, en las 9 variables, se obtuvo una mediana de 100 %, con una media de 100 % en 7 variables y 99.7 % en las dos restantes.

Por otra parte, para el problema I.6.2 del *benchmark* de Feynman, se decidió realizar un análisis gráfico de ciertas soluciones particulares de los algoritmos DSO y DSO-MO. En primer lugar, se representa gráficamente la ecuación original (Figura 6.5). Además, se muestra la solución con el menor RMSE que también cumple con la condición de ser no-creciente en θ, la cual fue obtenida a través del algoritmo DSO-MO (Figura 6.7).

Asimismo, para ambos algoritmos, se incluye la representación gráfica de las soluciones que satisfacen la propiedad de monotonicidad no-creciente en θ en la mínima proporción sobre el total de puntos en el conjunto de validación. Estas soluciones, que corresponden a las columnas *Min* de la Tabla 6.9, son consideradas las peores en cuanto a este criterio, ya que la solución de DSO tiene un 55.0 % de puntos que cumplen con la monotonicidad, mientras que la solución de DSO-MO presenta un 57.4 % de puntos con la misma propiedad. La solución correspondiente a DSO se puede observar en la Figura 6.6, mientras que la solución obtenida mediante DSO-MO se muestra en la Figura 6.8.

Como se puede observar, la solución correspondiente a la Figura 6.7 es la que más se asemeja gráficamente a la función objetivo y presenta el menor RMSE

en el conjunto de validación. Esta solución no solo se alinea con la forma de la función original, sino que también cumple con la condición de ser no-creciente en θ.

En contraste, las soluciones de las Figuras 6.6 y 6.8 no logran mantener la monotonicidad en un porcentaje significativo de puntos, con solo un 55.0 % y 57.4 % de puntos, respectivamente, que cumplen con esta propiedad. Esta falta de monotonicidad en ciertas regiones contribuye al incremento del RMSE en estas soluciones. Sin embargo, como fue reportado y mencionado anteriormente, ninguno de los algoritmos fue capaz de descubrir la ecuación correctamente.

Función	**DSO**			**DSO-MO**		
	Med	**IQR**	**Éxito**	**Med**	**IQR**	**Éxito**
Feynman						
I.6.2	**2.01e-02**	2.75e-03	×	2.24e-02	3.10e-03	×
I.9.18	8.76e-02	7.07e-03	×	**8.37e-02**	7.67e-03	×
I.32.17	3.11e+00	3.03e-01	×	**2.33e+00**	5.51e-01	×
I.41.16	**1.52e+00**	1.42e-01	×	1.58e+00	4.77e-01	×
II.6.15a	**1.93e-01**	1.81e-02	×	2.55e-01	2.92e-02	×
II.11.27	**1.14e-01**	5.71e-02	×	1.23e-01	4.99e-02	×
II.35.21	**1.89e+00**	1.59e-01	×	1.93e+00	2.07e-01	×
III.9.52	**1.21e+01**	1.47e-01	×	1.24e+01	6.07e-01	×
III.10.19	2.20e+00	4.01e-01	×	**2.17e+00**	4.56-01	×

Tabla 6.7: Mediana (Med), *Interquartile Range* (IQR), Mínimo (Min) y Máximo (Max) de la distribución de RMSE, junto a la indicación de éxito en la tarea de descubrir la ecuación original, y tiempo promedio en segundos (t_{avg}) para cada ecuación y algoritmo.

Función	**DSO**			**DSO MO**		
	Min	**Max**	t_{avg}	**Min**	**Max**	t_{avg}
Feynman						
I.6.2	**1.09e-02**	2.16e-02	1.88e+03	1.42e-02	2.41e-02	2.37e+03
I.9.18	**6.54e-02**	9.54e-02	1.75e+03	6.85e-02	9.18e-02	3.02e+03
I.32.17	2.10e+00	3.31e+00	2.07e+03	**1.59e+00**	3.74e+00	2.83e+03
I.41.16	**8.43e-01**	1.67e+00	2.00e+03	8.69e-01	2.05e+00	1.60e+03
II.6.15a	**1.53e-01**	2.16e-01	2.74e+03	1.95e-01	2.81e-01	1.81e+04
II.11.27	6.12e-02	1.88e-01	1.82e+03	**5.83e-02**	1.91e-01	2.50e+04
II.35.21	**1.48e+00**	2.09e+00	2.01e+03	1.52e+00	2.17e+00	2.14e+04
III.9.52	1.06e+01	1.25e+01	2.44e+03	**1.02e+01**	1.29e+01	1.20e+04
III.10.19	1.75e+00	2.64e+00	2.24e+03	**1.53e+00**	2.72e+00	4.86e+03

Tabla 6.8: Mínimo (Min), máximo (Max) de la distribución de RMSE y tiempo promedio en segundos (t_{avg}) para cada función y algoritmo.

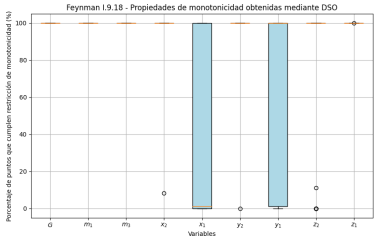

Figura 6.3: Boxplots que muestran el porcentaje de puntos que cumplen con las restricciones de monotonicidad para las 9 variables correspondientes a la ecuación Feynman I.9.18. Los resultados se obtuvieron tras 30 ejecuciones del algoritmo DSO.

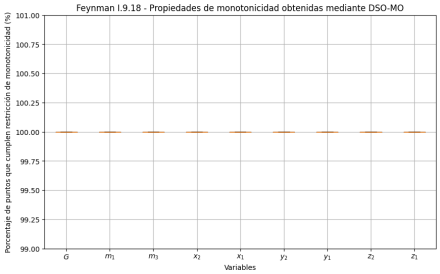

Figura 6.4: Boxplots que muestran el porcentaje de puntos que cumplen con las restricciones de monotonicidad para las variables correspondientes a la ecuación Feynman I.9.18. Los resultados se obtuvieron tras 30 ejecuciones del algoritmo DSO-MO.

Ecuación	**Variable**	**DSO**				**DSO - MO**			
		Media (%)	**Mediana** (%)	**IQR**	**Min**	**Media** (%)	**Mediana** (%)	**IQR**	**Min**
I.6.2	θ	96.9	100	0	55.0	98.5	100	0	**57.4**
I.9.18	G	100	100	0	**100**	99.7	100	0	91.8
	m_1	100	100	0	**100**	100	100	0	**100**
	m_2	100	100	0	**100**	100	100	0	**100**
	x_1	33.4	1.17	100	0	100	100	0	**100**
	y_1	55.8	100	98.7	0	100	100	0	**100**
	z_1	99.9	100	0	**100**	100	100	0	99.6
	x_2	96.9	100	0	8.3	99.7	100	0	**92.0**
	y_2	96.7	100	0	0	100	100	0	**100**
	z_2	77.0	100	0	0	100	100	0	**100**
I.32.17	ϵ	96.4	100	0	12.5	97.4	100	0	**33.3**
	c	99.2	100	0	77.4	100	100	0	**100**
	E_f	93.9	100	0	3.2	96.3	100	0	**43.5**
	r	91.7	100	0.3	**1.6**	94.6	100	0	0
	ω	91.9	100	0	8.7	98.5	100	0	**65.5**
	ω_0	93.2	100	0	2.7	99.7	100	0	**95.4**
I.41.16	T	96.5	100	0	**100**	100	100	0	99.5
	h	100	100	0	**100**	100	100	0	98.6
	k_b	100	100	0	**100**	100	100	0	**100**
	c	99.4	100	0	83.2	99.9	100	0	**99.5**

Ecuación	**Variable**	**DSO**				**DSO - MO**			
		Media (%)	**Mediana** (%)	**IQR**	**Min**	**Media** (%)	**Mediana** (%)	**IQR**	**Min**
II.6.15a	ϵ	88.5	100	2.4	0.1	100	100	0	**100**
	pd	96.4	100	0	38.1	100	100	0	**100**
	r	75.8	100	21.7	**0**	98.2	100	0	**0**
	x	100	100	0	**100**	99.3	100	0	81.2
	y	100	100	0	**100**	99.0	100	0	72.0
	z	94.4	100	0	42.2	98.2	100	0	**62.4**
II.11.27	n	100	100	0	**100**	99.6	100	0	87.9
	α	97.1	100	0	16.4	99.5	100	0	**89.3**
	ϵ	100	100	0	**100**	100	100	0	**100**
	E_f	100	100	0	**100**	98.6	100	0	58.0
II.35.21	n_ρ	92.9	100	3.1	**50.3**	96.8	100	0	49.1
	μ_M	91.7	100	3.8	**50.1**	91.7	100	0	26.3
	B	96.6	100	0	55.1	97.7	100	0	**76.0**
	k_b	97.1	100	0	**69.1**	91.3	100	0	0
	T	94.5	100	0	0	98.9	100	0	**75.6**
III.9.52	pd	100	100	0	**100**	99.9	100	0	98.5
	E_f	99.5	100	0	**84.6**	97.3	100	0	40.2
	h	99.8	100	0	**93.0**	90.4	100	0	0
III.10.19	μ_M	98.9	100	0	70.4	100	100	0	**100**
	B_x	100	100	0	**100**	90.1	100	0	50.9
	B_y	95.9	100	0	**2.6**	95.7	100	0	1.57
	B_z	98.2	100	0	48.8	93.6	100	0	**91.37**

Tabla 6.9: Media, Mediana, IQR y Mínimo del porcentaje de puntos del conjunto de validación que cumplen la restricción de monotonía por variable en las ecuaciones de Feynman.

Si se analizan nuevamente los resultados presentados en la Tabla 6.9, se observa que, para la ecuación III.9.52, el algoritmo DSO-MO arrojó resultados inferiores, ya que tanto la media como el valor mínimo disminuyeron para las tres variables con restricciones, aunque la mediana y el rango intercuartílico se mantuvieron en sus valores óptimos, 100 y 0, respectivamente. Para el resto de las ecuaciones, como fue mencionado anteriormente, además de obtener una mediana e IQR de 100 y 0, respectivamente, con DSO-MO, también se logra al menos una mejora en la media o el valor mínimo para al menos una de las variables.

Por otra parte, DSO-MO es más lento que DSO, como se refleja en los valores de t_{avg}, con tiempos promedio más altos en todas las funciones. Esto se debe a las modificaciones introducidas en DSO-MO, enfocadas en que se cumplan propiedades de monotonicidad para determinadas variables, implicando un costo computacional adicional al tener que manejar la dominancia de las distintas soluciones, debido al objetivo adicional que cuantifica el porcentaje de puntos de entrenamiento que no satisfacen tales propiedades.

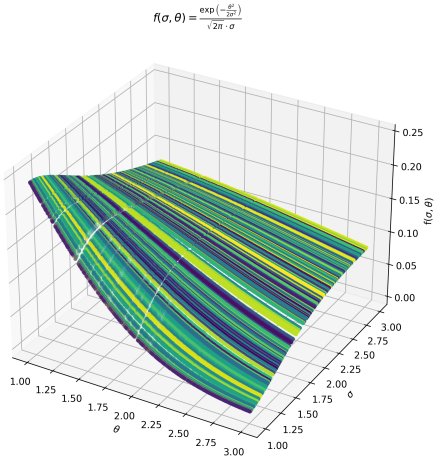

Figura 6.5: Representación gráfica de la función del *benchmark* Feynman I.6.2.

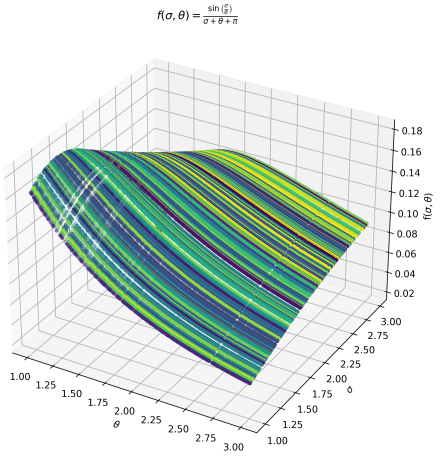

Figura 6.6: Solución obtenida por el algoritmo DSO que cumplió con la propiedad de monotonicidad no-creciente en θ en la menor cantidad de puntos para la función del *benchmark* Feynman I.6.2.

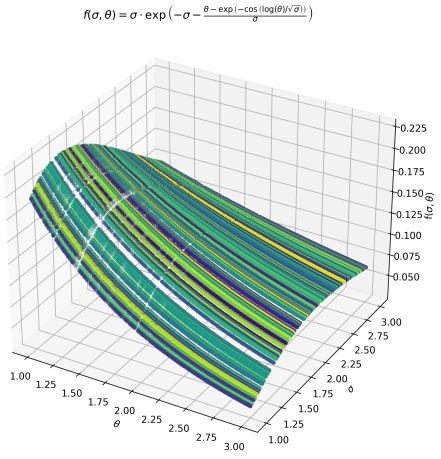

Figura 6.7: Solución obtenida por el algoritmo DSO-MO con menor RMSE y no-creciente en θ, para la función del *benchmark* Feynman I.6.2.

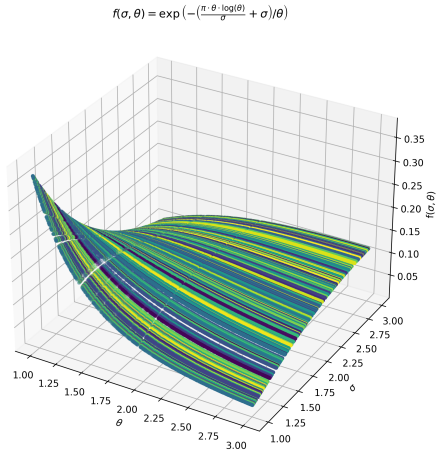

Figura 6.8: Solución obtenida por el algoritmo DSO-MO que cumplió con la propiedad de monotonicidad no-creciente en θ en la menor cantidad de puntos para la función del *benchmark* Feynman I.6.2.

Capítulo 7

Análisis experimental de DSO con operador de selección de Deb

En este capítulo se presenta la evaluación experimental realizada utilizando una variante del algoritmo DSO basada en el operador de selección propuesto por Deb, (2000). El método presentado por Deb, introducido en la Sección 3.2.7, propone un operador de selección que permite manejar restricciones con prioridades en los modelos. Este enfoque, en el que se privilegia el cumplimiento de las condiciones físicas sobre el error, se consideró adecuado de explorar debido a la cantidad de soluciones retornadas por DSO-MO que no cumplían las condiciones esperadas. Esto se integra en este proyecto a la hora de ejecutar el componente de PG de DSO. En la Sección 7.1 se describen las características de los experimentos, y en la Sección 7.2 se presentan y analizan los resultados obtenidos.

7.1. Configuración paramétrica

Para estos experimentos, se replicó el entorno de pruebas descrito previamente en la Sección 4.1.1, reutilizando los conjuntos de datos y los valores de hiperparámetros detallados en la Sección 6.3.

En el caso de los problemas de Feynman, el conjunto de problemas presentado en la Sección 6.2 fue reducido. Este subconjunto se conformó únicamente por aquellos casos en los que, tras 30 ejecuciones independientes de DSO-MO, se obtuvo al menos una solución donde para una variable no se cumplió la restricción de monotonía a lo largo de su rango. Se recuerda que en cada ejecución independiente, se seleccionó una solución de la población que representase un compromiso entre el cumplimiento de las restricciones y el error.

Por otro lado, se evaluaron nuevamente los problemas 1, 5, 7 y 8 de Vladis-

lavleva, específicamente para analizar las restricciones relacionadas con puntos indefinidos y valores no negativos, y poder comparar con los experimentos previos. Se reutilizaron los conjuntos de datos especificados en la Sección 4.4.3.

7.2. Resultados experimentales

7.2.1. Evaluación utilizando puntos indefinidos y RMSE

Al igual que en los experimentos reportados en la Sección 5.3, en la Tabla 7.1 se utiliza el símbolo ✓ en la columna *éxito* para señalar cuando un algoritmo logró obtener, en al menos una de sus 30 ejecuciones independientes, una expresión matemática equivalente a la original. También se considera exitoso si el algoritmo alcanza un error en el conjunto de entrenamiento igual a cero o muy cercano, aunque la expresión no sea exactamente equivalente a la buscada. En caso contrario, se emplea el símbolo × para denotar la falta de éxito. Además, la tabla reporta las medianas y los IQRs obtenidos.

Por otro lado, en la Tabla 7.2 se presentan los valores mínimos, máximos y el tiempo promedio de ejecución.

Función	**DSO-MO**			**DSO con Selección de Deb**		
	Med	**IQR**	**Éxito**	**Med**	**IQR**	**Éxito**
Vladislavleva 1	**1.26e-01**	2.26e-02	×	1.27e-01	2.52e-02	×
Vladislavleva 5	**5.90e-01**	1.11e-01	×	6.02e-01	1.61e-01	×
Vladislavleva 7	**4.20e+00**	2.73e-01	×	4.24e+00	2.32e-01	×
Vladislavleva 8	2.46e+00	9.07e+00	×	**2.20e+00**	5.67e+00	×

Tabla 7.1: Mediana (Med), rango intercuartílico (IQR) de la distribución de RMSE y éxito en la tarea de descubrir la ecuación original para cada función y algoritmo comparando DSO-MO con Selector de Deb.

Función	**DSO-MO**			**DSO con Selección de Deb**		
	Min	**Max**	t_{avg}	**Min**	**Max**	t_{avg}
Vladislavleva 1	**1.06e-01**	1.18e+02	2.39e+03	**1.06e-01**	4.79e+00	1.60e+03
Vladislavleva 5	5.06e-01	5.50e+00	2.50e+03	**3.41e-01**	7.10e-01	1.54e+03
Vladislavleva 7	**3.47e+00**	1.11e+02	1.69e+03	3.76e+00	4.66e+01	1.37e+03
Vladislavleva 8	**1.72e+00**	1.89e+02	2.18e+03	**1.72e+00**	2.15e+60	2.24e+03

Tabla 7.2: Mínimo (Min), máximo (Max) de la distribución de RMSE y tiempo promedio en segundos (t_{avg}) para cada función y algoritmo comparando DSO-MO con Selector de Deb.

En cuanto a las medianas del RMSE, esta variante no muestra una mejora significativa en comparación con el algoritmo DSO-MO, como se puede observar en la Tabla 7.2, donde la mediana del RMSE es menor en solamente 1 de 4 casos respecto a las obtenidas con DSO-MO. Por otro lado, el algoritmo que utiliza selección de Deb, muestra un IQR menor en las funciones 7 y 8 de Vladislavleva, lo que puede indicar una variabilidad menor en los resultados de las ejecuciones para las funciones más complejas o con errores más grandes. Respecto al valor mínimo de RMSE, solo se logró disminuir un caso. En cuanto al éxito al

Ecuación	**DSO-MO**	**DSO con Selección de Deb**
	RMSE=∞	**RMSE=∞**
Vladislavleva 1	11 [3,1,1,45,1,251,3,1,45,1,45]	**0**
Vladislavleva 5	0	**0**
Vladislavleva 7	2 [1,2]	**0**
Vladislavleva 8	3 [1,1,1]	**1[1]**

Tabla 7.3: Resumen de casos en los que RMSE es infinito o indefinido en 30 ejecuciones independientes para cada uno de los algoritmos (DSO-MO, Selector de Deb) para un subconjunto de las ecuaciones de Vladislavleva. Para el primer caso se indica cuántos puntos del *dataset* de validación provocan dicho comportamiento.

Ecuación	**DSO-MO**		**DSO con Selección de Deb**	
	RMSE=∞ con $x_i \neq 0$	**RMSE>100**	**RMSE=∞ con $x_i \neq 0$**	**RMSE>100**
Vladislavleva 1	2	2	**0**	**0**
Vladislavleva 5	**0**	0	**0**	**0**
Vladislavleva 7	2	0	**0**	**0**
Vladislavleva 8	3	**0**	**1**	3

Tabla 7.4: Resumen de casos en los que RMSE es infinito con $x_i \neq 0$, y RMSE mayor que 100 en 30 ejecuciones independientes para cada uno de los algoritmos (DSO-MO, Deb Selector) para un subconjunto de las ecuaciones de Vladislavleva. Para el primer caso se indica cuántos puntos del *dataset* de validación provocan dicho comportamiento.

hallar las funciones, se observa que el método sigue sin poder hallar ninguna representación equivalente a la original.

Por otro lado, en la Tabla 7.3 se puede observar que el número de casos en donde el RMSE es infinito, se reduce notablemente para todas las funciones evaluadas, siendo este cero para 3 de 4 funciones y solamente 1 ejecución presenta valores indefinidos en 1 punto en el conjunto de Validación. Para el caso de los RMSE mayores a 100, se puede observar que en la Tabla 7.4, la Función 1 presenta una reducción a 0 de estos casos, mientras que se presentan 3 nuevos en la función Vladislavleva 8. Se puede afirmar entonces que las soluciones encontradas presentan menor cantidad de puntos indefinidos en el conjunto de validación que las obtenidas mediante DSO-MO y DSO. Esto es un resultado esperable debido a que el operador de selección hace énfasis en guiar al algoritmo a un subespacio de soluciones que cumplen con las condiciones.

7.2.2. Evaluación utilizando *shape constraints* y RMSE

Al igual que en los experimentos reportados en la Sección 5.3, en la Tabla 7.5 se utiliza el símbolo ✓ en la columna *éxito* para señalar cuando un algoritmo logró obtener, en al menos una de sus 30 ejecuciones independientes, una expresión matemática equivalente a la original. También se considera exitoso si el algoritmo alcanza un error en el conjunto de entrenamiento igual a cero o muy cercano, aunque la expresión no sea exactamente equivalente a la buscada. En caso contrario, se emplea el símbolo × para denotar la falta de éxito. Además, la tabla reporta las medianas y los IQRs obtenidos.

Por otro lado, en la Tabla 7.6 se presentan los valores mínimos, máximos y el tiempo promedio de ejecución.

Función	DSO-MO			DSO con Selección de Deb		
	Med	IQR	Éxito	Med	IQR	Éxito
Feynman						
I.32.17	**2.33e+00**	5.51e-01	×	2.77e+00	3.92e-01	×
II.6.15a	2.55e-01	2.92e-02	×	**1.98e-01**	1.94e-02	×
II.35.21	1.93e+00	2.07e-01	×	**2.90e-01**	2.70e-02	×
III.9.52	1.24e+01	6.07e-01	×	**1.21e+01**	4.37e-01	×
III.10.19	**2.17e+00**	4.56e-01	×	2.26e+00	3.66e-01	×

Tabla 7.5: Mediana (Med), *Interquartile Range* (IQR), Mínimo (Min) y Máximo (Max) de la distribución de RMSE, junto a la indicación de éxito en la tarea de descubrir la ecuación original, y tiempo promedio en segundos (t_{avg}) para cada ecuación y algoritmo.

En cuanto a la mediana del RMSE, el algoritmo DSO con selección de Deb logra disminuir este valor en 3 de las 5 funciones evaluadas. En relación al IQR, se observa que dicha variante presenta valores menores que DSO-MO en 3 de las 5 funciones. Respecto al éxito en la tarea de descubrir la ecuación original, ambos algoritmos continúan sin lograrlo en ninguna de las ejecuciones.

Al analizar los tiempos promedio de ejecución (t_{avg}), se observa que el algoritmo DSO con selección de Deb reduce significativamente el tiempo requerido en la mayoría de los casos. Un ejemplo claro de esto se da en la función II.6.15a, donde el tiempo promedio de ejecución disminuye de 18100 segundos a 3980 segundos. Esta mejora se debe a que, en este caso, no se están ejecutando métodos particulares del algoritmo NSGA-II que requerían mayores tiempos de cómputo.

Por último, al analizar los datos de monotonicidad (ver Tabla 7.7), se evi-

Función	DSO-MO			DSO con Selección de Deb		
	Min	Max	t_{avg}	Min	Max	t_{avg}
Feynman						
I.32.17	**1.59e+00**	3.74e+00	2.83e+03	1.98e+00	3.28e+00	1.42e+03
II.6.15a	1.95e-01	2.81e-01	1.81e+04	**1.65e-01**	2.22e-01	3.98e+03
II.35.21	1.52e+00	2.17e+00	2.14e+04	**1.82e-01**	3.26e-01	3.05e+03
III.9.52	**1.02e+01**	1.29e+01	1.20e+04	1.06e+01	1.44e+01	1.11e+04
III.10.19	1.53e+00	2.72e+00	4.86e+03	**1.44e+00**	2.82e+00	4.55e+03

Tabla 7.6: Mínimo (Min), máximo (Max) de la distribución de RMSE y tiempo promedio en segundos (t_{avg}) para cada función y algoritmo.

dencia que la implementación con la selección de Deb en el algoritmo DSO, mejora significativamente el cumplimiento de las restricciones de monotonía en las soluciones generadas. En general, las soluciones producidas al aplicar dicho operador presentan medias más altas que DSO-MO para la mayoría de variables en todas las funciones, y en todas ellas, la mediana preserva el 100 %. Además, se observa un incremento notable en el mínimo porcentaje de cumplimiento de las restricciones de monotonía al utilizar el algoritmo DSO con selección de Deb en la gran mayoría de funciones, destacando III.10.19 en la que se obtiene que todas las soluciones generadas cumplen las restricciones de monotonicidad para todas las variables.

El problema I.6.2 del *benchmark* de Feynman no fue reportado en las tablas anteriores debido a que, en las soluciones provistas por el algoritmo DSO-MO, se observó que en todas las soluciones obtenidas la función fue no-creciente en θ en subintervalos que, en conjunto, cubrieron al menos el 50 % del rango de θ. Sin embargo, en línea con el análisis realizado en la Sección 6.4.2 para dicha ecuación, se realizó un análisis gráfico de ciertas soluciones particulares generadas por el algoritmo DSO con Selección de Deb. Todas las soluciones obtenidas en las 30 ejecuciones fueron no-crecientes en θ. En la Figura 7.1, se presenta la solución con el menor RMSE, la cual muestra una gran similitud con la representación original previamente mostrada en la Figura 6.5.

Por otro lado, en la Figura 7.2, se ilustra la solución obtenida por DSO con Selección de Deb que reportó el mayor error en entrenamiento. Este resultado podría deberse a que, en dicha ejecución, el algoritmo requirió un mayor número de iteraciones para alcanzar un subespacio de soluciones factibles, y solo posteriormente pudo seleccionar aquellas con menor error, lo que finalmente llevó a mejores soluciones en las demás ejecuciones.

Esto indica que el operador de selección de Deb es un método más eficaz a la hora de obtener soluciones que las cumplan, y esto se cumple dado que es más estricto a la hora de aplicar las restricciones definidas.

Ecuación	**Variable**	**DSO-MO**				**DSO con Selección de Deb**			
		Media (%)	**Mediana (%)**	**IQR**	**Min**	**Media (%)**	**Mediana (%)**	**IQR**	**Min**
I.32.17	ϵ	97.4	100	0	33.3	**99.9**	100	0	**96.1**
	c	**100**	100	0	**100**	99.4	100	0	81.7
	E_f	96.3	100	0	43.5	**98.6**	100	0	**67.9**
	r	94.6	100	0	0	**99.0**	100	0	**70.6**
	ω	**98.5**	100	0	65.5	**98.5**	100	0	**72.6**
	ω_0	**99.7**	100	0	**95.4**	98.6	100	0	57.6
II.6.15a	ϵ	**100**	100	0	**100**	**100**	100	0	97.8
	pd	**100**	100	0	**100**	**100**	100	0	99.4
	r	98.2	100	0	0	**99.9**	100	0	**94.8**
	x	99.3	100	0	81.2	**100**	100	0	**100**
	y	99.0	100	0	72.0	**99.1**	100	0	**73.1**
	z	98.2	100	0	62.4	**100**	100	0	**100**
II.35.21	n_ρ	96.8	100	0	49.1	**99.8**	100	0	**95.4**
	μ_M	91.7	100	0	26.3	**100**	100	0	**99.1**
	B	97.7	100	0	76.0	**100**	100	0	**99.1**
	k_b	91.3	100	0	0	**100**	100	0	**98.8**
	T	98.9	100	0	75.6	**100**	100	0	**100**
III.9.52	pd	**99.9**	100	0	**98.5**	97.7	100	0	55.6
	E_f	97.3	100	0	40.2	**99.8**	100	0	**96.0**
	h	90.4	100	0	0	**99.1**	100	0	**72.5**
III.10.19	μ_M	**100**	100	0	**100**	**100**	100	0	**100**
	B_x	90.1	100	0	50.9	**100**	100	0	**100**
	B_y	**100**	100	0	1.57	**100**	100	0	**100**
	B_z	93.6	100	0	91.37	**100**	100	0	**100**

Tabla 7.7: Media, Mediana, IQR y Mínimo del porcentaje de puntos del conjunto de validación que cumplen la restricción de monotonía por variable comparando DSO-MO con el DSO con Selección de Deb.

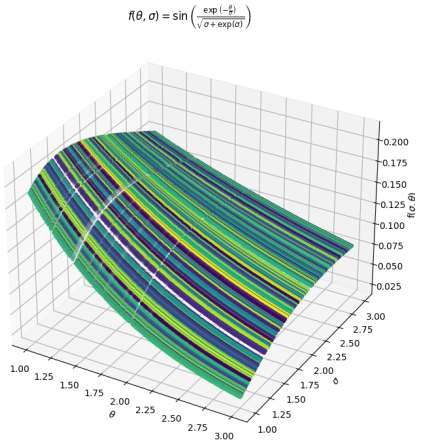

Figura 7.1: Solución obtenida por el algoritmo DSO con Selección de Deb con menor RMSE y no-creciente en θ, para la función del *benchmark* Feynman I.6.2.

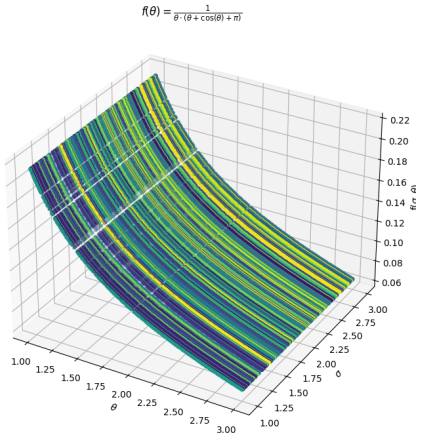

Figura 7.2: Solución obtenida por el algoritmo DSO con Selección de Deb que cumplió con la propiedad de monotonicidad no-creciente en θ en todos los puntos para la función del *benchmark* Feynman I.6.2, pero presentó el mayor error en los puntos de entrenamiento.

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se sintetizan los resultados obtenidos en el proyecto y se presentan las conclusiones derivadas del mismo, evaluando tanto los logros alcanzados como las dificultades encontradas. Se destacan los aportes realizados y se proponen posibles extensiones, identificando áreas de investigación futura para mejorar la solución desarrollada o profundizar en aspectos específicos. Además, se incluye una autocrítica sobre el desarrollo del trabajo, considerando las lecciones aprendidas y oportunidades de mejora.

8.1. Conclusiones

El proyecto cumplió progresivamente con los objetivos planteados, partiendo de una revisión exhaustiva de técnicas de inteligencia computacional para la generación de modelos subrogados. El marco teórico proporcionó una base sólida, explorando conceptos como la RS, la PG y las redes neuronales aplicadas a estas tareas, incluyendo enfoques híbridos. En particular, se prestó especial atención a los modelos que combinan programación genética y aprendizaje profundo, destacando métodos como *Deep Symbolic Optimization*, el cual, en cada iteración, emplea redes neuronales para la generación inicial de la población en PG.

A partir del relevamiento inicial, se seleccionaron tres algoritmos representativos: DSO, DSR y Operon, considerando sus características paradigmáticas y enfoques contrastantes. DSR se centra en un modelo basado exclusivamente en redes neuronales, mientras que Operon emplea PG, y DSO combina ambos enfoques de manera híbrida. Estos algoritmos fueron escogidos no solo por sus fundamentos técnicos, sino también porque representan el estado del arte en sus categorías según el *benchmark* SRBench, reconocido como referencia en el campo. Los resultados de SRBench, especialmente en problemas reales, vinculados varios de ellos a la ingeniería de procesos, demostraron que estos algoritmos

son capaces de obtener errores bajos y menores requerimientos computacionales respecto a otros algoritmos.

Posteriormente a dicha selección, se llevó a cabo una fase de experimentación siguiendo un enfoque monoobjetivo, basándonos únicamente en la minimización del error de los modelos generados. En dicha fase, se evaluó el desempeño de los algoritmos DSO, DSR y Operon, aplicando pruebas estadísticas sobre distintos *benchmarks*. Los problemas abordados incluían tanto casos del mundo real, representativos de aplicaciones prácticas en el contexto de ingeniería de procesos, como problemas sintéticos diseñados específicamente para evaluar el desempeño de los algoritmos en escenarios de distintos grados de dificultad. Para analizar la significancia de los resultados, se utilizó el Test de Friedman como prueba general (*ómnibus*), la cual permitió determinar si existían diferencias significativas en la distribución de la raíz del error cuadrático medio entre los algoritmos. Los resultados obtenidos indicaron que Operon mostró un rendimiento estadísticamente superior respecto a DSO y DSR en la mayoría de las ecuaciones evaluadas, lo que fue verificado mediante el procedimiento *post-hoc* de Holm. Este análisis permitió rechazar la hipótesis nula de igualdad de medianas del RMSE con un nivel de confianza del 95 %, concluyendo que Operon logró medianas de RMSE más bajas. No obstante, en lo que respecta a generar soluciones con la menor cantidad de puntos indefinidos, Operon fue el peor de los tres algoritmos para el *benchmark* de Vladislavleva, seguido por DSO.

Cabe destacar que no fue posible replicar los resultados reportados en el *benchmark* SRBench (2022) que indican que DSO era el mejor algoritmo, lo cual sugiere la influencia de factores adicionales, como omisiones en los valores utilizados para los distintos hiperparámetros disponibles, omisiones en detalles que deben ser ajustados en los algoritmos, los conjuntos de datos utilizados, entre otros. Sin embargo, se optó por utilizar DSO para la fase de diseño de la variante multiobjetivo debido a su buen rendimiento en dicho *benchmark*, y a que es un algoritmo híbrido que combina redes neuronales y PG, lo que resultaba de particular interés para las líneas de investigación de nuestros tutores.

Posteriormente a los experimentos previos, se diseñó e implementó un algoritmo multiobjetivo, denominado DSO-MO, cuyo objetivo principal era poder introducir restricciones adicionales sobre las propiedades que deberían seguir idealmente las funciones generadas por los algoritmos. Estas restricciones incluyeron propiedades de monotonicidad para algunas variables, y la no-negatividad de las imágenes. Además, esto también permite reducir la presencia de puntos indefinidos en las soluciones generadas. Dichos puntos indefinidos se referían a situaciones en las que los valores en las soluciones eran indeterminados, como en el caso de los valores negativos que eran interpretados como indefinidos.

Durante el proceso de implementación del algoritmo DSO-MO, se presentó una dificultad técnica relacionada con la incompatibilidad entre los *frameworks* *DEAP* y jMetalPy, lo que impidió una implementación directa en jMetalPy. Como solución a este problema, se descompuso la estructura del algoritmo NSGA-II dentro de jMetalPy, lo que resultó en la arquitectura detallada en la Sección 4.2.1.2.

Los experimentos realizados fueron evaluados bajo dos aproximaciones: por

un lado, minimizar el error (RMSE) mientras se impone restricciones de monotonicidad; y por otro lado, minimizar el RMSE mientras se intentaba minimizar a su vez la cantidad de puntos indefinidos, que incluían valores negativos interpretados como indefinidos. Esta experimentación se llevó a cabo en el contexto de dos subconjuntos de problemas, el *benchmark* de Feynman (para la optimización con *shape constraints*) y el de Vladislavleva (para la optimización con puntos indefinidos), como se detalla en el capítulo 6.

En la evaluación de DSO-MO, incorporando el objetivo de minimizar los puntos indefinidos, los resultados mostraron que DSO-MO no logró mejoras significativas en términos del RMSE para la mayoría de las funciones evaluadas. Aunque en algunos casos DSO-MO presentó una mediana menor del RMSE, esto fue acompañado de un aumento en la variabilidad (IQR), lo que indica resultados menos consistentes. Los valores mínimos y máximos del RMSE tampoco muestran una ventaja clara: DSO-MO logró reducir el valor mínimo solo en la mitad de las funciones.

Respecto al éxito en la recuperación de expresiones originales, ninguno de los algoritmos logró recuperar una expresión equivalente a la original en ninguna de las 30 ejecuciones independientes, al igual que como sucedió con los algoritmos monoobjetivo. En relación a los puntos indefinidos en el conjunto de validación, DSO-MO mostró una frecuencia similar en comparación con DSO en la mayoría de las ecuaciones. Esto indica que la estrategia multiobjetivo de DSO-MO, enfocada en minimizar los puntos indefinidos, no siempre fue efectiva y, en ciertos casos, incluso incrementó la presencia de valores indefinidos. Además, se observó que muchos puntos indefinidos no estaban relacionados con valores de $x_i = 0$ (no existían puntos con estas características en el conjunto de entrenamiento). Sin embargo, tales puntos indefinidos se encontraban siempre fuera del rango de entrenamiento, lo que sugiere que la extrapolación fuera del rango de entrenamiento supone un desafío para el algoritmo.

Por otro lado, DSO-MO presentó un mayor costo computacional en comparación con DSO debido a la inclusión del objetivo adicional para minimizar la cantidad de puntos indefinidos. Sin embargo, en las dos ecuaciones donde DSO había reportado RMSE mayores a 100, DSO-MO logró reducir a cero la cantidad de ejecuciones con errores mayores a tal valor, lo que muestra una mayor efectividad en el control del error para estos casos.

Este análisis sugiere que, aunque DSO-MO introduce mejoras para manejar puntos indefinidos, su mayor variabilidad y el incremento en el tiempo de cómputo limitan sus ventajas en términos de rendimiento global.

Al evaluar el desempeño de DSO-MO considerando la minimización del error junto con las restricciones de monotonía, los resultados muestran un comportamiento mixto en comparación con DSO. En términos de error, DSO-MO no logra superar consistentemente a DSO, con mejoras observadas solo en algunos casos. Además, aunque el algoritmo DSO-MO busca garantizar el cumplimiento de las restricciones de monotonía, este cumplimiento no siempre se traduce en una reducción significativa del error, lo que refleja el compromiso entre mantener propiedades de monotonía y minimizar el RMSE.

Por otra parte, el análisis de los frentes de Pareto revela que, en muchos

experimentos, se obtuvieron soluciones con error bajo que cumplían las restricciones, lo cual es un comportamiento deseado. Sin embargo, hubo ocasiones donde las soluciones con menor error no eran completamente monótonas, indicando que permitir cierta flexibilidad en las restricciones podría ser beneficioso para reducir el error, como se ha reportado en estudios previos.

En cuanto al éxito en la recuperación de expresiones originales, ninguno de los algoritmos evaluados logró encontrar la forma exacta de las ecuaciones, lo que sugiere una alta complejidad de las funciones de prueba evaluadas. Adicionalmente, para estos objetivos DSO-MO vuelve a presentar un incremento en el tiempo de cómputo debido al manejo adicional de restricciones, lo cual representa un costo computacional a tener en cuenta.

En resumen, aunque las *shape constraints*, como la monotonicidad y la no-negatividad, son enfoques que se hallaron útiles para guiar la búsqueda de soluciones en problemas de RS, la minimización del error en muchos casos predomina sobre el cumplimiento de dichas restricciones. Esto resalta la necesidad de explorar métodos adicionales o ajustes en el algoritmo para balancear de manera más efectiva la optimización del error y el cumplimiento de las *shape constraints*. Como se observó en los experimentos, el operador de selección de Deb podría considerarse en conjunto a otros métodos y estrategias de manejo de soluciones no factibles, debido a que fue notoriamente el método más efectivo a la hora de satisfacer las propiedades físicas. Estos resultados también reafirman lo observado en trabajos previos (Kronberger y cols., 2021), donde la minimización del error a menudo compromete las propiedades deseadas en los modelos generados.

Es importante recalcar que en nuestra opinión el *benchmark* de Vladislavleva debería ser considerado como uno de los principales de referencia, ya que múltiples trabajos han reportado resultados poco alentadores debido a la dificultad que impone validar con puntos fuera del rango de entrenamiento. Este desafío es particularmente relevante en el contexto de ingeniería de procesos, donde las variabilidades en las condiciones pueden llevar a escenarios fuera del rango de entrenamiento. Por lo tanto, sería interesante que nuevos métodos incluyan su rendimiento en este *benchmark*, ya que proporciona un marco de referencia importante para la evaluación de algoritmos en contextos industriales donde tales variabilidades son comunes.

Por último, se observa que no todas las variables fueron incluidas en las soluciones generadas por los algoritmos, probablemente debido a que algunas de ellas introducían poca variabilidad al ser modificadas. Este hallazgo sugiere que en ciertos casos, las variables con menor impacto no se seleccionan en las soluciones finales, lo que puede ser un aspecto importante en la interpretación de los resultados.

8.2. Autocrítica y gestión del proyecto

El desarrollo del proyecto presentó diversas dificultades, especialmente en la integración de distintas bibliotecas y en la implementación de nuevas funcionalidades. La falta de recursos y el gran tiempo requerido para llevar a cabo los

experimentos limitó la posibilidad de realizar pruebas adicionales que podrían haber permitido una validación más exhaustiva de los resultados. La gestión del proyecto se complicó debido a los desafíos técnicos y a la necesidad de realizar ajustes constantes en la implementación. Sin embargo, se logró cumplir con los objetivos principales y se dejaron sentadas las bases para futuras investigaciones.

En retrospectiva, se recomienda priorizar una mayor planificación en la fase de integración de herramientas y considerar un enfoque modular para facilitar el desarrollo y la experimentación. Asimismo, sería conveniente destinar más tiempo a la validación de los resultados y a la comparación con estudios previos para fortalecer las conclusiones obtenidas.

8.3. Trabajo futuro

A partir de los resultados obtenidos en este estudio, se han identificado varias direcciones prometedoras para futuras investigaciones. Estas líneas de trabajo, que quedaron fuera del alcance del presente proyecto debido a limitaciones de tiempo y recursos, presentan oportunidades para mejorar y extender los enfoques utilizados.

En primer lugar, una extensión interesante sería la incorporación de propiedades funcionales adicionales, como la concavidad o convexidad, lo cual podría enriquecer el proceso de búsqueda. De manera similar a como se implementaron restricciones de monotonía, considerar restricciones adicionales podría guiar mejor la exploración del espacio de soluciones y aumentar la precisión en la identificación de expresiones simbólicas. Además, se podría evaluar la incorporación de hasta tres objetivos diferentes en la optimización multiobjetivo, permitiendo un enfoque más holístico que equilibre tanto el ajuste de los datos como las propiedades estructurales de las expresiones.

Otra extensión relevante sería evaluar el algoritmo multiobjetivo en el *framework* Operon, que ya proporciona herramientas robustas para definir operadores personalizados y realizar optimización multiobjetivo mediante su implementación de NSGA-II. Esta exploración permitiría comparar y validar los resultados obtenidos con DSO-MO, sirviendo como un análisis complementario o incluso como un posible contraejemplo para profundizar las conclusiones finales.

En cuanto a la optimización multiobjetivo, una estrategia interesante sería la ponderación de objetivos mediante umbrales específicos. Este enfoque permitiría ajustar la influencia de cada objetivo según su importancia relativa, mejorando el balance entre el ajuste a los datos y el cumplimiento de las restricciones físicas o estructurales. Este tipo de ponderación podría proporcionar una manera más controlada de guiar la búsqueda de soluciones.

Un área de investigación que podría explorarse es el uso de modelos avanzados basados en *Transformers*, como Symformer, introducido en la Sección 2.4.3, o grandes modelos de lenguaje (LLMs), los cuales han mostrado resultados prometedores en la generación y descubrimiento de expresiones simbólicas. Sin embargo, estos modelos presentan un desafío significativo en términos de capacidad computacional, lo que fue una barrera en el presente proyecto. Esto es

especialmente relevante cuando se considera que, para problemas con más de dos variables, sería necesario reentrenar los modelos, ya que actualmente están entrenados solo para una o dos variables, lo que implica tiempos de entrenamiento muy largos. Además, a medida que el número de variables aumenta, el espacio de soluciones posibles se amplía considerablemente, lo que probablemente incrementaría aún más los tiempos de entrenamiento si se pretende lograr resultados igual de buenos. Superar estas limitaciones computacionales podría abrir nuevas oportunidades para generar modelos simbólicos de mayor calidad e interpretabilidad.

Finalmente, creemos que es necesario seguir explorando enfoques alternativos para el manejo de soluciones no factibles dentro del contexto multiobjetivo. Una estrategia a considerar es la eliminación directa de estas soluciones de la población, en lugar de simplemente aplicar penalizaciones. Adicionalmente, el uso de penalizaciones más severas podría forzar un mayor cumplimiento de las propiedades deseadas, como la monotonía, mejorando así la calidad de las soluciones generadas.

Referencias

Abolafia, D. A., Norouzi, M., Shen, J., Zhao, R., y Le, Q. V. (2018). Neural program synthesis with priority queue training. *arXiv preprint arXiv:1801.03526*.

Acampora, G., Kaymak, U., Loia, V., y Vitiello, A. (2013, October). Applying nsga-ii for solving the ontology alignment problem. En *Proceedings of the 2013 ieee international conference on systems, man and cybernetics (smc 2013).* (Available from: https://www.researchgate.net/figure/The-general-structure-of-NSGA-II_fig1_271417039 [accessed 3 Aug 2024]) doi: 10.1109/SMC.2013.191

Ahn, S., Kim, J., Lee, H., y Shin, J. (2020). Guiding deep molecular optimization with genetic exploration. En *Proceedings of the conference on neural information processing systems (neurips).*

Aldeia, G., y de França, F. (2022). Interpretability in symbolic regression: a benchmark of explanatory methods using the feynman data set. *Genetic Programming and Evolvable Machines*, *23*, 309–349. doi: 10.1007/s10710-022-09435-x

Ali, S., Abuhmed, T., El-Sappagh, S., y Muhammad, K. (2023). Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence. *Information Fusion*, *101*. Descargado de https://www.researchgate.net/publication/370111593 doi: 10.1016/j.inffus.2023.101805

Arnaldo, I., Krawiec, K., y O'Reilly, U.-M. (2014). Multiple regression genetic programming. En *Proceedings of the 2014 annual conference on genetic and evolutionary computation (gecco)* (pp. 879–886).

Benítez-Hidalgo, A., Nebro, A. J., García-Nieto, J., Oregi, I., y Del Ser, J. (2019, April 17). jmetalpy: a python framework for multi-objective optimization with metaheuristics. *arXiv preprint arXiv:1903.02915*.

Bladek, I., y Krawiec, K. (2019). Solving symbolic regression problems with formal constraints. En *Proceedings of the genetic and evolutionary computation conference* (pp. 977–984).

Box, G. E. P., y Draper, N. R. (1986). *Empirical model-building and response surfaces*. New York, NY, USA: John Wiley & Sons, Inc.

Bruneton, J.-P., Cazenille, L., Douin, A., y Reverdy, V. (2019). Exploration and exploitation in symbolic regression using quality-diversity and evolutio-

nary strategies algorithms. *arXiv preprint arXiv:1906.03959*. Descargado de https://arxiv.org/abs/1906.03959

Burlacu, B. (2023). Gecco'2022 symbolic regression competition: Post-analysis of the operon framework. En *Proceedings of the companion conference on genetic and evolutionary computation* (p. 2412–2419). New York, NY, USA: Association for Computing Machinery. Descargado de https://doi.org/10.1145/3583133.3596390 doi: 10.1145/3583133.3596390

Burlacu, B., Kronberger, G., y Kommenda, M. (2019). Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, *20*(4), 467–498. doi: 10.1007/s10710-019-09371-3

Burlacu, B., Kronberger, G., Kommenda, M., y Affenzeller, M. (2019). Parsimony measures in multi-objective genetic programming for symbolic regression. En *Proceedings of the genetic and evolutionary computation conference companion* (p. 338–339). New York, NY, USA: Association for Computing Machinery. Descargado de https://doi.org/10.1145/3319619.3322087 doi: 10.1145/3319619.3322087

Castillo, F., Marshall, K., Green, J., y Kordon, A. (2003). A methodology for combining symbolic regression and design of experiments to improve empirical model building. En *Genetic and evolutionary computation conference* (pp. 1975–1985).

Castillo, F. A., Villa, C. M., y Kordon, A. K. (2013). Symbolic regression model comparison approach using transmitted variation. En *Genetic programming theory and practice x* (pp. 139–154).

Cava, W. L., Singh, T. R., Taggart, J., Suri, S., y Moore, J. H. (2019). Learning concise representations for regression by evolving networks of trees. En *International conference on learning representations (iclr)*.

Cava, W. L., Spector, L., y Danai, K. (2016). Epsilon-lexicase selection for regression. En *Proceedings of the genetic and evolutionary computation conference 2016 (gecco '16)* (pp. 741–748). New York, NY, USA: ACM. doi: 10.1145/2908812.2908898

Celis, O. S., Cuyt, A., y Verdonk, B. (2007). Rational approximation of vertical segments. *Numerical Algorithms*, *45*, 375–388.

Chakraborty, A., Sivaram, A., Samavedham, L., y Venkatasubramanian, V. (2020). Mechanism discovery and model identification using genetic feature extraction and statistical testing. *Computers & Chemical Engineering*, *140*, 106900.

Chakraborty, A., Sivaram, A., y Venkatasubramanian, V. (2021). Ai-darwin: a first principles-based model discovery engine using machine learning. *Computers & Chemical Engineering*, *154*, 107470.

Cherkassky, V., y Mulier, F. (1998). *Learning from data: Concepts, theory, and methods*. New York, NY, USA: John Wiley & Sons, Inc.

Choi, K., Youn, B., y Yang, R.-J. (2002). Moving least squares method for reliability-based design optimization. En L. et al. (Ed.), *Genetic programming theory and practice ii* (cap. 17). Ann Arbor, MI, USA: Springer.

Coello Coello, C. A., Lamont, G. B., y Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems* (2nd ed.). New York: Springer.

Computing, S., y Group, I. I. S. R. (s.f.). *Statistical inference in computational intelligence and data mining.* https://sci2s.ugr.es/sicidm#Nonparametric%20Tests. (Accedido: 30 de septiembre de 2024)

Cuyt, A., y Verdonk, B. (1985). Multivariate rational interpolation. *Computing*, *34*, 41–61.

Da Silva, F. L., Goncalves, A., y cols. (2023). Language model-accelerated deep symbolic optimization. *Neural Computing and Applications*, 1–17.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, *186*(2), 311-338. Descargado de https://www.sciencedirect.com/science/article/pii/S0045782599003898 doi: https://doi.org/10.1016/S0045-7825(99)00389-8

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms.* Chichester: Wiley.

Deb, K., Pratap, A., Agarwal, S., y Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. on Evolutionary Computation*, *6*(2), 182–197.

de França, F. O. (2018). A greedy search tree heuristic for symbolic regression. *Information Sciences*, *442*, 18–32.

de França, F. O., Virgolin, M., Kommenda, M., Majumder, M. S., Cranmer, M., Espada, G., ... La Cava, W. G. (2023, Jul). Interpretable symbolic regression for data science: Analysis of the 2022 competition. *arXiv preprint arXiv:2304.01117*. Descargado de https://arxiv.org/abs/2304.01117

Derner, E., Kubalík, J., Ancona, N., y Babuška, R. (2020). Constructing parsimonious analytic models for dynamic systems via symbolic regression. *Applied Soft Computing*, *94*, 106432. Descargado de https://www.sciencedirect.com/science/article/pii/S1568494620303720 doi: https://doi.org/10.1016/j.asoc.2020.106432

Derrac, J., Garcıa, S., Molina, D., y Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput*, *1*(1), 3–18.

Diqi Chen, Y. W., y Gao, W. (2020). Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning. *Applied Intelligence*, *50*.

Drichel, A. (2008). *Búsqueda en anchura - breadth-first search tree.* Descargado de https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_anchura#/media/Archivo:Breadth-first-tree.svg (Imagen disponible en Wikipedia)

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, *56*, 52–64.

Faris, J., Hayes, C., Gonçalves, A., Sprenger, K., Faissol, D., Petersen, B., ... Silva, F. (2024). Pareto front training for multi-objective symbolic opti-

mization. *Journal Name*.

Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, *13*(2), 87–129.

Ferreira, J. (2022). *Algoritmos evolutivos para el aprendizaje de modelos en la industria de procesos* (PhD Thesis). Universidad de la República, Facultad de Ingeniería, Montevideo, Uruguay.

Ferreira, J., Torres, A., y Pedemonte, M. (2023). A kaizen programming algorithm for multi-output regression based on a heterogeneous island model. *Neural Computing and Applications*, *35*, 1-19. doi: 10.1007/s00521-023-08335-0

Feynman, R. P., Leighton, R. B., y Sands, M. (1964). *The feynman lectures on physics* (Vol. 1-3). Reading, MA, USA: Addison-Wesley. (3 volumes)

Floreano, D., Dürr, P., y Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, *1*, 47–62.

Flórez, C. A. C., Bolaños, R. A., y Cabrera, A. M. (2008, Septiembre). Algoritmo multiobjetivo nsga-ii aplicado al problema de la mochila. *Scientia et Technica*, *XIV*(39), 206. (Fecha de Recepción: 4 de Junio de 2008. Fecha de Aceptación: 30 de Julio de 2008.)

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., y Gagné, C. (2012, Jul). Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, *19*(1), 1–67.

Gangwani, T., y Peng, J. (2018). Policy optimization by genetic distillation. En *Proceeding of the international conference on learning representations (iclr)*.

Goldberg, D. E., y Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine Learning*, *3*(2), 95–99. Descargado de https://doi.org/10.1023/A:1022602019183 doi: 10.1023/A:1022602019183

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. Descargado de http://www.deeplearningbook.org

Haykin, S. (1994). *Neural networks: A comprehensive foundation*. New York, NY, USA: Macmillan College Publishing Company, Inc.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, *6*, 65–70.

Hornby, G. S. (2006). Alps: The age-layered population structure for reducing the problem of premature convergence. En *Proceedings of the 8th annual conference on genetic and evolutionary computation (gecco '06)* (pp. 815–822). New York, NY, USA: ACM. doi: 10.1145/1143997.1144142

Igel, C., y Kreutz, M. (1999). Using fitness distributions to improve the evolution of learning structures. En *Proceedings of the 1999 congress on evolutionary computation-cec99 (cat. no. 99th8406)* (Vol. 3, p. 1902–1909). IEEE.

Johnson, R. A., y Wichern, D. W. (1988). *Applied multivariate statistical analysis* (2nd ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Khadka, S., y Tumer, K. (2018). Evolution-guided policy gradient in reinforcement learning. En *Proceeding of the conference on neural information*

processing systems (neurips).

Kim, S., Lu, P.-Y., Mukherjee, S., Gilbert, M., Jing, L., Ceperic, V., y Soljacic, M. (2019). Integration of neural network-based symbolic regression in deep learning for scientific discovery.

Kleijnen, J. P. (2008). Kriging metamodeling in simulation: A review. *European Journal of Operational Research*.

Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. *MIT Press*.

Koza, J. R. (1994). *Genetic programming ii: Automatic discovery of reusable programs*. Cambridge, Massachusetts: MIT Press.

Kronberger, G., de Franca, F. O., Burlacu, B., Haider, C., y Kommenda, M. (2021, March). Shape-constrained symbolic regression—improving extrapolation with prior knowledge. *Evolutionary Computation*, *30*(1), 75–98. (Manuscript received: 16 December 2019; revised: 27 July 2020, 27 October 2020, 22 February 2021, and 9 March 2021; accepted: 26 March 2021.) doi: 10.1162/evco_a_00294

La Cava, W., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., ... Moore, J. H. (2021, Jul). Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*. Descargado de https://arxiv.org/abs/2107.14351

Landajuela, M., Lee, C. S., Yang, J., y cols. (2022). A unified framework for deep symbolic regression. En *Advances in neural information processing systems* (Vol. 35, pp. 33985–33998).

Landajuela, M., Petersen, B. K., Kim, S. K., Santiago, C. P., Glatt, R., Mundhenk, T. N., ... Faissol, D. M. (2021, Jul). Improving exploration in policy gradient search: Application to symbolic optimization. *arXiv preprint arXiv:2107.09158*.

Levin, D. (1998). The approximation power of moving least-squares. *Mathematical Computation*, *67*(224), 1517–1531.

Li, H., Waschkowski, F., Zhao, Y., y Sandberg, R. D. (2023). Turbulence model development based on a novel method combining gene expression programming with an artificial neural network. *arXiv*. (http://arxiv.org/abs/2301.07293)

Li, L., Fan, M., Singh, R., y Riley, P. (2019). Neural-guided symbolic regression with semantic prior. *arXiv preprint*. (Retrieved from arXiv:1901.07714)

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2016). Continuous control with deep reinforcement learning. En *Proceedings of the international conference on learning representations (iclr)*.

Lin, H. W., Tegmark, M., y Rolnick, D. (2017). What does it mean to be a deep learning model? *Journal of Statistical Physics*, *168*, 1223.

Liu, D., Virgolin, M., Alderliesten, T., y Bosman, P. A. N. (2022). Evolvability degeneration in multi-objective genetic programming for symbolic regression. *arXiv preprint arXiv:2202.06983*. Descargado de https://arxiv.org/abs/2202.06983

Ljung, L. (2010, April). Perspectives on system identification. *Annual Reviews in Control*, *34*(1), 1–12.

Long, Z., Lu, Y., y Dong, B. (2018). Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *arXiv preprint arXiv:1812.04426*. Descargado de http://arxiv.org/abs/1812.04426

Long, Z., Lu, Y., Ma, X., y Dong, B. (2018). Pde-net: Learning pdes from data. En *Proceedings of the 35th international conference on machine learning (icml)* (pp. 3208–3216). Descargado de http://proceedings.mlr.press/v80/long18a.html

Lu, P. Y., Kim, S., y Soljačić, M. (2019). Extracting interpretable physical parameters from spatiotemporal systems using unsupervised learning. *arXiv preprint arXiv:1907.06011*. Descargado de http://arxiv.org/abs/1907.06011

Lüders, B., Schläger, M., Korach, A., y Risi, S. (2017). Continual and one-shot learning through neural networks with dynamic external memory. En *European conference on the applications of evolutionary computation* (pp. 886–901).

Makke, N., y Chawla, S. (2024). Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, *57*(2). Descargado de https://doi.org/10.1007/s10462-023-10622-0 doi: 10.1007/s10462-023-10622-0

Martius, G., y Lampert, C. H. (2016). Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*. Descargado de http://arxiv.org/abs/1610.02995

Matt J Kusner, B. P., y Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. En *Proceedings of the 34th proceeding of the international conference on machine learning (icml)-volume 70* (p. 1945–1954). JMLR. org.

McConaghy, T. (2011). Ffx: Fast, scalable, deterministic symbolic regression technology. En *Genetic programming theory and practice ix* (pp. 235–260).

Mejía, R. I. C. (2023). *Regresión simbólica mediante redes neuronales* (Tesis para optar al grado de Magíster en Ciencias de la Ingeniería, mención Eléctrica, y memoria para optar al título de Ingeniero Civil Eléctrico). Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ingeniería Eléctrica, Santiago de Chile.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization, and constraints. En L. Eshelman (Ed.), *Proceedings of the sixth international conference on genetic algorithms* (pp. 151–158). San Mateo: Morgan Kauffman.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., ... Hodjat, B. (2019). *Evolving deep neural networks*. Artificial Intelligence in the Age of Neural Networks and Brain Computing.

Miller, J. F., y Harding, S. L. (2008). Cartesian genetic programming. , 2701–2726.

Mitchell, M. (1996). *An introduction to genetic algorithms*. The MIT Press.

Montastruc, L., Azzaro-Pantel, C., Pibouleau, L., y Domenech, S. (2004). Use of genetic algorithms and gradient based optimization techniques for calcium phosphate precipitation. *Chemical Engineering and Processing: Process Intensification*, *43*(10), 1289–1298. Descargado de https://www.sciencedirect.com/science/article/pii/S0255270103002733 doi: 10.1016/j.cep.2003.12.002

Moraglio, A., Krawiec, K., y Johnson, C. G. (2012). Geometric semantic genetic programming. , 21–31.

Mozafari-Kermani, M., Sur-Kolay, S., Raghunathan, A., y Jha, N. K. (2015). Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE Journal of Biomedical and Health Informatics*, *19*(6), 1893–1905. doi: 10.1109/JBHI.2014.2344095

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., y Petersen, B. K. (2021, Nov). Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*. Descargado de https://arxiv.org/abs/2111.00053

Navarro, D. (2023). *Learning statistics with r: A tutorial for psychology students and other beginners*. University of New South Wales. (Compiled on 12/17/2023)

Oltean, M., y Grosan, C. (2003). A comparison of several linear genetic programming techniques. *Complex Systems*, *14*(4), 285–314.

O'Neill, M., y Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, *5*(4), 349–358.

Pawlak, T., y Krawiec, K. (2018). Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. *Evolutionary Computation*, *26*(2), 177–212.

Pedemonte, M., Luna, F., y Alba, E. (2018). A theoretical and empirical study of the trajectories of solutions on the grid of systolic genetic search. *Inf Sci*, *445*, 97–117.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., y Kim, J. T. (2019). Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*. Descargado de https://doi.org/10.48550/arXiv.1912.04871 (Published at International Conference on Learning Representations, 2021)

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., y Liao, Q. (2016). *Theory i: Why and when can deep networks avoid the curse of dimensionality?* (CBMM Memo Series; 058). Center for Brains, Minds and Machines (CBMM). Descargado de http://hdl.handle.net/1721.1/105443 (arXiv:1611.00740v5, formerly titled "Why and When Can Deep – but Not Shallow – Networks Avoid the Curse of Dimensionality: a Review
)

Pourchot, A., y Sigaud, O. (2019). Cem-rl: Combining evolutionary and gradient-based methods for policy search. En *Proceedings of the international conference on learning representations (iclr)*.

Powell, D., y Skolnick, M. M. (1993). Using genetic algorithms in engineering

design optimization with nonlinear constraints. En S. Forrest (Ed.), *Proceedings of the fifth international conference on genetic algorithms* (pp. 424–430). San Mateo: Morgan Kauffman.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: A review. En J. C. Mason y M. G. Cox (Eds.), *Algorithms for approximation* (pp. 143–167). Oxford: Clarendon Press.

Praksova, R. (2011). Eureqa: Software review. *Genetic Programming and Evolvable Machines*, *12*, 173–178.

Qi, C. R., Su, H., Mo, K., y Guibas, L. J. (2017). PointNet: Deep learning on point sets for 3d classification and segmentation. En *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 652–660).

Qiong Chen, Q. X. H. W., Mengxing Huang, y Wang, J. (2020). Reinforcement learning-based genetic algorithm in optimizing multidimensional data discretization scheme. *Mathematical Problems in Engineering*.

Real, E., Liang, C., So, D. R., y Le, Q. V. (2019). Automl-zero: Evolving machine learning algorithms from scratch. En *Proceedings of the international conference on machine learning (icml)*.

Richardson, J. T., Palmer, M. R., Liepins, G., y Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. En J. D. Schaffer (Ed.), *Proceedings of the third international conference on genetic algorithms* (pp. 191–197). San Mateo: Morgan Kauffman.

Risi, S., y Togelius, J. (2017). Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*, 25–41.

Ruberto, S., Terragni, V., y Moore, J. H. (2020). Sgp-dt: Semantic genetic programming based on dynamic targets. En T. Hu, N. Lourenço, E. Medvet, y F. Divina (Eds.), *Genetic programming* (pp. 167–183). Cham: Springer International Publishing.

Rudin, C. (2019, May). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206–215. Descargado de https://doi.org/10.1038/s42256-019-0048-x doi: 10.1038/s42256-019-0048-x

Sacks, J., Welch, W., Mitchell, T. J., y Wynn, H. (1989). Design and analysis of computer experiments. *Statistical Science*, *4*, 409–435.

Sahoo, S. S., Lampert, C. H., y Martius, G. (2018). Learning equations for extrapolation and control. *arXiv preprint arXiv:1806.07259*. Descargado de http://arxiv.org/abs/1806.07259

Schmidt, M., y Lipson, H. (2008, December). Coevolution of fitness predictors. *IEEE Transactions on Evolutionary Computation*, *12*(6), 736–749. doi: 10.1109/TEVC.2008.919006

Schmidt, M., y Lipson, H. (2011). Age-fitness pareto optimization. En *Genetic programming theory and practice viii* (pp. 129–146). Springer.

Schmidt, M. D., y Lipson, H. (2009). Incorporating expert knowledge in evolutionary search: A study of seeding methods. En *Proceedings of the 11th annual conference on genetic and evolutionary computation* (pp. 1091–1098).

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*. Descargado de https://doi.org/10.48550/arXiv.1912.05911 (Cite as: arXiv:1912.05911 [cs.LG] (or arXiv:1912.05911v1 [cs.LG] for this version))

Schnur, J. J., y Chawla, N. V. (2023). Information fusion via symbolic regression: A tutorial in the context of human health. *Information Fusion*, 326–335. doi: 10.1016/j.inffus.2022.11.030

Seborg, D. E., Edgar, T. F., Mellichamp, D. A., y Doyle, F. J. I. (2016). *Process dynamics and control* (4th ed.). New York: John Wiley & Sons.

Sehgal, A., La, H., Louis, S., y Nguyen, H. (2019). Deep reinforcement learning using genetic algorithm for parameter optimization. En *Proceedings of the ieee international conference on robotic computing (icrc).*

Sheskin, D. J. (2011). *Handbook of parametric and nonparametric statistical procedures* (5th ed.). Boca Raton: Chapman and Hall/CRC.

Simyung Chang, J. C., John Yang, y Kwak, N. (2018). Genetic-gated networks for deep reinforcement learning. En *Proceeding of the conference on neural information processing systems (neurips).*

Smits, G. F., y Kotanchek, M. (2005). Pareto-front exploitation in symbolic regression. , 283–299.

Srinivas, N., y Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, *2*(3), 221–248.

Stanley, K. O., Clune, J., Lehman, J., y Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*.

Stanley, K. O., y Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*, 99–127.

Stephens, T. (2015). *gplearn: Genetic programming in python.* https://gplearn.readthedocs.io/en/stable/intro.html. (Accedido el 24 de setiembre del 2024)

Stewart, R., y Ermon, S. (2017). Label-free supervision of neural networks with physics and domain knowledge. En *Thirty-first aaai conference on artificial intelligence* (pp. 2576–2582).

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., y Clune, J. (2018). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv:1712.06567*.

Sun, H., y Moscato, P. (2019). A memetic algorithm for symbolic regression. En *2019 ieee congress on evolutionary computation* (pp. 2167–2174).

Tian, Y., Wang, Q., Huang, Z., Li, W., Dai, D., Yang, M., ... Fink, O. (2020). Off-policy reinforcement learning for efficient and effective gan architecture search. En *Proceedings of the european conference on computer vision (eccv).*

Topchy, A., y Punch, W. (2001). Faster genetic programming based on local gradient search of numeric leaf values. En *Proceedings of the third annual conference on genetic and evolutionary computation (gecco).*

Toropov, V. (1989). Simulation approach to structural optimization. *Structural Optimization*, *1*(1), 37–46.

Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., y Blunsom, P. (2018). Neural arithmetic logic units. En *Advances in neural information processing systems (neurips)* (pp. 8035–8044). Descargado de http://papers.nips.cc/paper/8027-neural-arithmetic-logic-units.pdf

Udrescu, S.-M., y Tegmark, M. (2020). *Ai feynman: a physics-inspired method for symbolic regression.* Descargado de https://arxiv.org/abs/1905.11481

Valipour, M., You, B., Panju, M., y Ghodsi, A. (2021). *Symbolicgpt: Transformer-based language model for symbolic regression.* arXiv preprint arXiv:2106.14131. Descargado de https://arxiv.org/abs/2106.14131

Vapnik, V. (1997). The support vector method. En *Icann '97: Proceedings of the 7th international conference on artificial neural networks* (pp. 263–271). London, UK: Springer-Verlag.

Vastl, M., Kulhánek, J., Kubalík, J., Derner, E., y Babuška, R. (2024, March). Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access*. (Received 21 January 2024, accepted 24 February 2024, date of publication 7 March 2024, date of current version 15 March 2024) doi: 10.1109/ACCESS.2024.3374649

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Jones, L. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2023). *Attention is all you need.* Descargado de https://arxiv.org/abs/1706.03762

Versino, D., Tonda, A., y Bronkhorst, C. A. (2017). Data driven modeling of plastic deformation. *Computer Methods in Applied Mechanics and Engineering*, *318*, 981–1004.

Virgolin, M., Alderliesten, T., y Bosman, P. A. N. (2019). Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. En *Proceedings of the genetic and evolutionary computation conference (gecco '19)* (pp. 1084–1092).

Virgolin, M., Alderliesten, T., Witteveen, C., y Bosman, P. A. N. (2020). Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary Computation*, tba.

Vladislavleva, E. J., Smits, G. F., y den Hertog, D. (2009). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, *13*(2), 333–349. doi: 10.1109/TEVC.2008.927706

Wierstra, D., Schaul, T., Peters, J., y Schmidhuber, J. (2008). Natural evolution strategies. En *Proceedings of the ieee congress on evolutionary computation.*

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3-4), 229–256.

Worm, T., y Chiu, K. (2013). Prioritized grammar enumeration: Symbolic regression by dynamic programming. En *Proceedings of the 15th annual conference on genetic and evolutionary computation* (pp. 1021–1028).

Wu, N. L., Wang, X. Y., Ge, T., Wu, C., y Yang, R. (2017). Parametric identification and structure searching for underwater vehicle model using symbolic regression. *Journal of Marine Science and Technology (Japan)*, *22*(1), 51–60. doi: 10.1007/s00773-016-0396-8

Yukang Chen, Q. Z. S. X. C. H. L. M., Gaofeng Meng, y Wang, X. (2019). Renas: Reinforced evolutionary neural architecture search. En *Proceeding of the ieee conference on computer vision and pattern recognition (cvpr).*

Zhang, M., y Smart, W. (2004). Genetic programming with gradient descent search for multiclass object classification. En M. Keijzer, U.-M. O'Reilly, S. Lucas, E. Costa, y T. Soule (Eds.), *Genetic programming* (pp. 399–408). Springer Berlin Heidelberg.

Zheng, D., Luo, V., Wu, J., y Tenenbaum, J. B. (2018). Unsupervised learning of latent physical properties using perception-prediction networks. En *Proceedings of the 34th conference on uncertainty in artificial intelligence (uai)* (pp. 497–507).

Anexo A

A.1. Funciones de Feynman

Function and Terminal Set	Operaciones y Constantes
FeynmanA	$+$, $-$, $\times$, $\div$, $\neg$, $\sqrt{}$, e, π, log, inv, cos, sin
FeynmanB	$+$, $-$, $\times$, $\div$, 0, $\neg$
FeynmanC	$+$, $-$, $\times$, $\div$, $\neg$, $\sqrt{}$, e, π, log, inv, cos, abs, arcsin, arctan, sin, 0

Tabla A.1: Operaciones en los conjuntos de funciones Feynman. Destacar que el operador $\neg$ es un operador unario y representa la negación (en los Reales, el opuesto), mientras que el operador $-$ es de aridad 2 y representa la sustracción.

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
I.6.2a	$\frac{e^{-\theta^2/2}}{\sqrt{2\pi}}$	$\theta : U[1,3]$	Feynman-A
I.6.2	$\frac{e^{-(\frac{\theta^2}{2\sigma^2})}}{\sqrt{2\pi}\sigma}$	$\sigma : U[1,3]$ $\theta : U[1,3]$	Feynman-A
I.6.2b	$\frac{e^{-(\frac{(\theta-\theta_1)^2}{2\sigma^2})}}{\sqrt{2\pi}\sigma}$	$\sigma : U[1,3]$ $\theta : U[1,3]$ $\theta_1 : U[1,3]$	Feynman-A
I.8.14	$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$	$x_1 : U[1,5]$ $x_2 : U[1,5]$ $y_1 : U[1,5]$ $y_2 : U[1,5]$	Feynman-A
I.9.18	$\frac{G\cdot m_1\cdot m_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$	$m_1 : U[1,2]$ $m_2 : U[1,2]$ $G : U[1,2]$ $x_1 : U[3,4]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$x_2 : U[1, 2]$ $y_1 : U[3, 4]$ $y_2 : U[1, 2]$ $z_1 : U[3, 4]$ $z_2 : U[1, 2]$	
I.10.7	$\frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}$	$m_0 : U[1, 5]$ $v : U[1, 2]$ $c : U[3, 10]$	Feynman-A
I.11.19	$x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3$	$x_1 : U[1, 5]$ $x_2 : U[1, 5]$ $x_3 : U[1, 5]$ $y_1 : U[1, 5]$ $y_2 : U[1, 5]$ $y_3 : U[1, 5]$	Feynman-A
I.12.1	$\mu \cdot N_n$	$\mu : U[1, 5]$ $N_n : U[1, 5]$	Feynman-A
I.12.2	$\frac{q_1 \cdot q_2}{4\pi\epsilon r^2}$	$q_1 : U[1, 5]$ $q_2 : U[1, 5]$ $\epsilon : U[1, 5]$ $r : U[1, 5]$	Feynman-A
I.12.4	$\frac{q_1}{4\pi\epsilon r^2}$	$q_1 : U[1, 5]$ $\epsilon : U[1, 5]$ $r : U[1, 5]$	Feynman-A
I.12.5	$q_2 \cdot E_f$	$q_2 : U[1, 5]$ $E_f : U[1, 5]$	Feynman-A
I.12.11	$q \cdot (E_f + B \cdot v \cdot \sin(\theta))$	$q : U[1, 5]$ $E_f : U[1, 5]$ $B : U[1, 5]$ $v : U[1, 5]$ $\theta : U[1, 5]$	Feynman-A
I.13.4	$\frac{1}{2}m(v^2 + u^2 + w^2)$	$m : U[1, 5]$ $v : U[1, 5]$ $u : U[1, 5]$ $w : U[1, 5]$	Feynman-A
I.13.12	$G \cdot m_1 \cdot m_2 \cdot \left(\frac{1}{r_2} - \frac{1}{r_1}\right)$	$m_1 : U[1, 5]$ $m_2 : U[1, 5]$ $r_1 : U[1, 5]$ $r_2 : U[1, 5]$ $G : U[1, 5]$	Feynman-A
I.14.3	$m \cdot g \cdot z$	$m : U[1, 5]$ $g : U[1, 5]$ $z : U[1, 5]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
I.14.4	$\frac{1}{2}k_{\text{spring}} \cdot x^2$	$k_{\text{spring}} : U[1, 5]$ $x : U[1, 5]$	Feynman-A
I.15.3x	$\frac{x-u \cdot t}{\sqrt{1-\frac{u^2}{c^2}}}$	$x : U[5, 10]$ $u : U[1, 2]$ $c : U[3, 20]$ $t : U[1, 2]$	Feynman-C
I.15.3t	$\frac{t-\frac{u \cdot x}{c^2}}{\sqrt{1-\frac{u^2}{c^2}}}$	$x : U[1, 5]$ $c : U[3, 10]$ $u : U[1, 2]$ $t : U[1, 5]$	Feynman-B
I.15.1	$\frac{m_0 \cdot v}{\sqrt{1-\frac{v^2}{c^2}}}$	$m_0 : U[1, 5]$ $v : U[1, 2]$ $c : U[3, 10]$	Feynman-A
I.16.6	$\frac{u+v}{1+\frac{u \cdot v}{c^2}}$	$u : U[1, 5]$ $v : U[1, 5]$ $c : U[3, 10]$	Feynman-A
I.18.4	$\frac{m_1 \cdot r_1 + m_2 \cdot r_2}{m_1+m_2}$	$m_1 : U[1, 5]$ $m_2 : U[1, 5]$ $r_1 : U[1, 5]$ $r_2 : U[1, 5]$	Feynman-A
I.18.12	$r \cdot F \cdot \sin(\theta)$	$r : U[1, 5]$ $F : U[1, 5]$ $\theta : U[0, 5]$	Feynman-A
I.18.14	$m \cdot r \cdot v \cdot \sin(\theta)$	$m : U[1, 5]$ $r : U[1, 5]$ $v : U[1, 5]$ $\theta : U[1, 5]$	Feynman-A
I.24.6	$\frac{1}{2}m(\omega^2 + \omega_0^2) \cdot \frac{1}{2}x^2$	$m : U[1, 3]$ $\omega : U[1, 3]$ $\omega_0 : U[1, 3]$ $x : U[1, 3]$	Feynman-A
I.25.13	$\frac{q}{C}$	$q : U[1, 5]$ $C : U[1, 5]$	Feynman-A
I.26.2	$\arcsin(n \cdot \sin(\theta_2))$	$n : U[0, 1]$ $\theta_2 : U[1, 5]$	Feynman-A
I.27.6	$\frac{1}{\frac{1}{d_1}+\frac{n}{d_2}}$	$d_1 : U[1, 5]$ $d_2 : U[1, 5]$ $n : U[1, 5]$	Feynman-A
I.29.4	$\frac{\omega}{c}$	$\omega : U[1, 10]$ $c : U[1, 10]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
I.29.16	$\sqrt{x_1^2 + x_2^2 - 2 \cdot x_1 \cdot x_2 \cdot \cos(\theta_1 - \theta_2)}$	$x_1 : U[1,5]$ $x_2 : U[1,5]$ $\theta_1 : U[1,5]$ $\theta_2 : U[1,5]$	Feynman-A
I.30.3	$\frac{\text{Int}_0 \cdot \sin(n \cdot \theta/2)^2}{\sin(\theta/2)^2}$	$\text{Int}_0 : U[1,5]$ $n : U[1,5]$ $\theta : U[1,5]$	Feynman-A
I.30.5	$\arcsin\left(\frac{\lambda}{n \cdot d}\right)$	$\lambda : U[1,2]$ $n : U[1,5]$ $d : U[2,5]$	Feynman-A
I.32.5	$\frac{q^2 \cdot a^2}{6\pi\epsilon c^3}$	$q : U[1,5]$ $a : U[1,5]$ $\epsilon : U[1,5]$ $c : U[1,5]$	Feynman-A
I.32.17	$\frac{1}{2}\epsilon c E_f^2 \cdot \frac{8\pi r^2}{3} \cdot \frac{\omega^4}{(\omega^2 - \omega_0^2)^2}$	$\epsilon : U[1,2]$ $c : U[1,2]$ $E_f : U[1,2]$ $r : U[1,2]$ $\omega : U[1,2]$ $\omega_0 : U[3,5]$	Feynman-A
I.34.8	$\frac{q \cdot v \cdot B}{p}$	$q : U[1,5]$ $v : U[1,5]$ $B : U[1,5]$ $p : U[1,5]$	Feynman-A
I.34.1	$\frac{\omega_0}{1 - \frac{v}{c}}$	$\omega_0 : U[1,5]$ $v : U[1,2]$ $c : U[3,10]$	Feynman-A
I.34.14	$\frac{1 + \frac{v}{c}}{\sqrt{1 - \frac{v^2}{c^2}}} \cdot \omega_0$	$\omega_0 : U[1,5]$ $v : U[1,2]$ $c : U[3,10]$	Feynman-A
I.34.27	$\frac{h}{2\pi} \cdot \omega$	$h : U[1,5]$ $\omega : U[1,5]$	Feynman-A
I.37.4	$I_1 + I_2 + 2 \cdot \sqrt{I_1 \cdot I_2} \cdot \cos(\delta)$	$I_1 : U[1,5]$ $I_2 : U[1,5]$ $\delta : U[1,5]$	Feynman-A
I.38.12	$\frac{4\pi\epsilon \cdot \left(\frac{h}{2\pi}\right)^2}{m \cdot q^2}$	$\epsilon : U[1,5]$ $h : U[1,5]$ $m : U[1,5]$ $q : U[1,5]$	Feynman-A
I.39.10	$\frac{3}{2} \cdot p_F \cdot V$	$p_F : U[1,5]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$V : U[1, 5]$	
I.39.11	$\frac{1}{\gamma-1} \cdot p_F \cdot V$	$\gamma : U[2, 5]$ $p_F : U[1, 5]$ $V : U[1, 5]$	Feynman-A
I.39.22	$\frac{n \cdot k_b \cdot T}{V}$	$n : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$ $V : U[1, 5]$	Feynman-A
I.40.1	$n_0 \cdot e^{-\frac{m \cdot g \cdot x}{k_b \cdot T}}$	$n_0 : U[1, 5]$ $m : U[1, 5]$ $g : U[1, 5]$ $x : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-A
I.41.16	$\frac{\frac{h}{2\pi} \cdot \omega^3}{\pi^2 c^2 \cdot \left(e^{\frac{h}{2\pi} \cdot \frac{\omega}{k_b \cdot T}} - 1\right)}$	$\omega : U[1, 5]$ $T : U[1, 5]$ $h : U[1, 5]$ $k_b : U[1, 5]$ $c : U[1, 5]$	Feynman-A
I.43.16	$\frac{\mu_{\text{drift}} \cdot q \cdot \text{Volt}}{d}$	$\mu_{\text{drift}} : U[1, 5]$ $q : U[1, 5]$ $\text{Volt} : U[1, 5]$ $d : U[1, 5]$	Feynman-A
I.43.31	$\mu_e \cdot k_b \cdot T$	$\mu_e : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-A
I.43.43	$\frac{1}{\gamma-1} \cdot k_b \cdot \frac{v}{A}$	$\gamma : U[2, 5]$ $k_b : U[1, 5]$ $v : U[1, 5]$ $A : U[1, 5]$	Feynman-A
I.44.4	$n \cdot k_b \cdot T \cdot \ln\left(\frac{V_2}{V_1}\right)$	$n : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$ $V_1 : U[1, 5]$ $V_2 : U[1, 5]$	Feynman-A
I.47.23	$\sqrt{\gamma \cdot \frac{p_r}{\rho}}$	$\gamma : U[1, 5]$ $p_r : U[1, 5]$ $\rho : U[1, 5]$	Feynman-A
I.48.2	$\frac{m \cdot c^2}{\sqrt{1 - \frac{v^2}{c^2}}}$	$m : U[1, 5]$ $v : U[1, 2]$	Feynman-B

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$c : U[3, 10]$	
I.50.26	$x_1 \cdot \left(\cos(\omega \cdot t) + \alpha \cdot \cos(\omega \cdot t)^2\right)$	$x_1 : U[1, 3]$ $\omega : U[1, 3]$ $t : U[1, 3]$ $\alpha : U[1, 3]$	Feynman-A
II.2.42	$\frac{\kappa \cdot (T_2 - T_1) \cdot A}{d}$	$\kappa : U[1, 5]$ $T_1 : U[1, 5]$ $T_2 : U[1, 5]$ $A : U[1, 5]$ $d : U[1, 5]$	Feynman-A
II.3.24	$\frac{\mathrm{P}}{4\pi r^2}$	$\mathrm{P} : U[1, 5]$ $r : U[1, 5]$	Feynman-A
II.4.23	$\frac{q}{4\pi\epsilon r}$	$q : U[1, 5]$ $\epsilon : U[1, 5]$ $r : U[1, 5]$	Feynman-A
II.6.11	$\frac{p_d \cdot \cos(\theta)}{4\pi\epsilon r^2}$	$\epsilon : U[1, 3]$ $p_d : U[1, 3]$ $\theta : U[1, 3]$ $r : U[1, 3]$	Feynman-A
II.6.15a	$\frac{p_d \cdot 3z \cdot \sqrt{x^2+y^2}}{4\pi\epsilon r^5}$	$\epsilon : U[1, 3]$ $p_d : U[1, 3]$ $x : U[1, 3]$ $y : U[1, 3]$ $z : U[1, 3]$ $r : U[1, 3]$	Feynman-A
II.6.15b	$\frac{p_d \cdot 3 \cdot \cos(\theta) \cdot \sin(\theta)}{4\pi\epsilon r^3}$	$\epsilon : U[1, 3]$ $p_d : U[1, 3]$ $\theta : U[1, 3]$ $r : U[1, 3]$	Feynman-A
II.8.7	$\frac{\frac{3}{5} \cdot q^2}{4\pi\epsilon d}$	$q : U[1, 5]$ $\epsilon : U[1, 5]$ $d : U[1, 5]$	Feynman-A
II.8.31	$\frac{\epsilon \cdot E_f^2}{2}$	$\epsilon : U[1, 5]$ $E_f : U[1, 5]$	Feynman-A
II.10.9	$\frac{\sigma_{\mathrm{den}}}{\epsilon \cdot (1+\chi)}$	$\sigma_{\mathrm{den}} : U[1, 5]$ $\epsilon : U[1, 5]$ $\chi : U[1, 5]$	Feynman-A
II.11.3	$\frac{q \cdot E_f}{m \cdot (\omega_0^2 - \omega^2)}$	$q : U[1, 3]$ $E_f : U[1, 3]$ $m : U[1, 3]$ $\omega_0 : U[3, 5]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$\omega : U[1, 2]$	
II.11.17	$n_0 \cdot \left(1 + \frac{p_d \cdot E_f \cdot \cos(\theta)}{k_b \cdot T}\right)$	$n_0 : U[1, 3]$ $p_d : U[1, 3]$ $E_f : U[1, 3]$ $k_b : U[1, 3]$ $T : U[1, 3]$ $\theta : U[1, 3]$	Feynman-A
II.11.20	$\frac{n_\rho \cdot p_d^2 \cdot E_f}{3 \cdot k_b \cdot T}$	$n_\rho : U[1, 5]$ $p_d : U[1, 5]$ $E_f : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-A
II.11.27	$\frac{n \cdot \alpha}{1 - \frac{n \cdot \alpha}{3}} \cdot \epsilon \cdot E_f$	$n : U[0, 1]$ $\alpha : U[0, 1]$ $\epsilon : U[1, 2]$ $E_f : U[1, 2]$	Feynman-A
II.11.28	$1 + \frac{n \cdot \alpha}{1 - \frac{n \cdot \alpha}{3}}$	$n : U[0, 1]$ $\alpha : U[0, 1]$	Feynman-A
II.13.17	$\frac{2 \cdot I}{4\pi\epsilon \cdot c^2 \cdot r}$	$\epsilon : U[1, 5]$ $c : U[1, 5]$ $I : U[1, 5]$ $r : U[1, 5]$	Feynman-A
II.13.23	$\frac{\rho_{c_0}}{\sqrt{1 - \frac{v^2}{c^2}}}$	$\rho_{c_0} : U[1, 5]$ $v : U[1, 2]$ $c : U[3, 10]$	Feynman-A
II.13.34	$\frac{\rho_{c_0} \cdot v}{\sqrt{1 - \frac{v^2}{c^2}}}$	$\rho_{c_0} : U[1, 5]$ $v : U[1, 2]$ $c : U[3, 10]$	Feynman-A
II.15.4	$-\mu_M \cdot B \cdot \cos(\theta)$	$\mu_M : U[1, 5]$ $B : U[1, 5]$ $\theta : U[1, 5]$	Feynman-A
II.15.5	$-p_d \cdot E_f \cdot \cos(\theta)$	$p_d : U[1, 5]$ $E_f : U[1, 5]$ $\theta : U[1, 5]$	Feynman-A
II.21.32	$\frac{q}{4\pi\epsilon r(1 - \frac{v}{c})}$	$q : U[1, 5]$ $\epsilon : U[1, 5]$ $r : U[1, 5]$ $v : U[1, 2]$ $c : U[3, 10]$	Feynman-A
II.24.17	$\sqrt{\frac{\omega^2}{c^2} - \frac{\pi^2}{d^2}}$	$\omega : U[4, 6]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$c : U[1, 2]$ $d : U[2, 4]$	
II.27.16	$\epsilon \cdot c \cdot E_f^2$	$\epsilon : U[1, 5]$ $c : U[1, 5]$ $E_f : U[1, 5]$	Feynman-A
II.27.18	$\epsilon \cdot E_f^2$	$\epsilon : U[1, 5]$ $E_f : U[1, 5]$	Feynman-A
II.34.2a	$\frac{q \cdot v}{2\pi r}$	$q : U[1, 5]$ $v : U[1, 5]$ $r : U[1, 5]$	Feynman-A
II.34.2	$\frac{q \cdot v \cdot r}{2}$	$q : U[1, 5]$ $v : U[1, 5]$ $r : U[1, 5]$	Feynman-A
II.34.11	$\frac{g_* \cdot q \cdot B}{2 \cdot m}$	$g_* : U[1, 5]$ $q : U[1, 5]$ $B : U[1, 5]$ $m : U[1, 5]$	Feynman-A
II.34.29a	$\frac{q \cdot h}{4\pi m}$	$q : U[1, 5]$ $h : U[1, 5]$ $m : U[1, 5]$	Feynman-A
II.34.29b	$\frac{g_* \cdot \mu_M \cdot B \cdot J_z}{h/(2\pi)}$	$g_* : U[1, 5]$ $\mu_M : U[1, 5]$ $B : U[1, 5]$ $J_z : U[1, 5]$ $h : U[1, 5]$	Feynman-A
II.35.18	$\frac{n_0}{e^{\frac{\mu_M \cdot B}{k_b \cdot T}} + e^{-\frac{\mu_M \cdot B}{k_b \cdot T}}}$	$n_0 : U[1, 3]$ $\mu_M : U[1, 3]$ $B : U[1, 3]$ $k_b : U[1, 3]$ $T : U[1, 3]$	Feynman-A
II.35.21	$n_\rho \cdot \mu_M \cdot \tanh\left(\frac{\mu_M \cdot B}{k_b \cdot T}\right)$	$n_\rho : U[1, 5]$ $\mu_M : U[1, 5]$ $B : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-C
II.36.38	$\mu_M \cdot \frac{B}{k_b \cdot T} + \frac{\mu_M \cdot \alpha}{\epsilon \cdot c^2 \cdot k_b \cdot T} \cdot M$	$\mu_M : U[1, 3]$ $B : U[1, 3]$ $k_b : U[1, 3]$ $T : U[1, 3]$ $\alpha : U[1, 3]$ $\epsilon : U[1, 3]$ $c : U[1, 3]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$M : U[1, 3]$	
II.37.1	$\mu_M \cdot (1 + \chi) \cdot B$	$\mu_M : U[1, 5]$ $B : U[1, 5]$ $\chi : U[1, 5]$	Feynman-A
II.38.3	$\frac{Y \cdot A \cdot x}{d}$	$Y : U[1, 5]$ $A : U[1, 5]$ $x : U[1, 5]$ $d : U[1, 5]$	Feynman-A
II.38.14	$\frac{Y}{2 \cdot (1+\sigma)}$	$Y : U[1, 5]$ $\sigma : U[1, 5]$	Feynman-A
III.4.32	$\frac{1}{e^{\frac{h}{2\pi} \cdot \frac{\omega}{k_b \cdot T}} - 1}$	$h : U[1, 5]$ $\omega : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-A
III.4.33	$\frac{\frac{h}{2\pi} \cdot \omega}{e^{\frac{h}{2\pi} \cdot \frac{\omega}{k_b \cdot T}} - 1}$	$h : U[1, 5]$ $\omega : U[1, 5]$ $k_b : U[1, 5]$ $T : U[1, 5]$	Feynman-A
III.7.38	$\frac{2 \cdot \mu_M \cdot B}{\frac{h}{2\pi}}$	$\mu_M : U[1, 5]$ $B : U[1, 5]$ $h : U[1, 5]$	Feynman-A
III.8.54	$\sin\left(\frac{E_n \cdot t}{\frac{h}{2\pi}}\right)^2$	$E_n : U[1, 2]$ $t : U[1, 2]$ $h : U[1, 4]$	Feynman-A
III.9.52	$\frac{p_d \cdot E_f \cdot t}{\frac{h}{2\pi}} \cdot \frac{\sin\left(\frac{(\omega-\omega_0) \cdot t}{2}\right)^2}{\left(\frac{(\omega-\omega_0) \cdot t}{2}\right)^2}$	$p_d : U[1, 3]$ $E_f : U[1, 3]$ $t : U[1, 3]$ $h : U[1, 3]$ $\omega : U[1, 5]$ $\omega_0 : U[1, 5]$	Feynman-A
III.10.19	$\mu_M \cdot \sqrt{B_x^2 + B_y^2 + B_z^2}$	$\mu_M : U[1, 5]$ $B_x : U[1, 5]$ $B_y : U[1, 5]$ $B_z : U[1, 5]$	Feynman-A
III.12.43	$n \cdot \frac{h}{2\pi}$	$n : U[1, 5]$ $h : U[1, 5]$	Feynman-A
III.13.18	$\frac{2 \cdot E_n \cdot d^2 \cdot k}{\frac{h}{2\pi}}$	$E_n : U[1, 5]$ $d : U[1, 5]$	Feynman-A

Nombre	**Ecuación**	**Espacio de entradas**	***Function-Terminal Set***
		$k : U[1, 5]$ $h : U[1, 5]$	
III.14.14	$I_0 \cdot \left(e^{\frac{q \cdot \text{Volt}}{k_b \cdot T}} - 1\right)$	$I_0 : U[1, 5]$ $q : U[1, 2]$ $\text{Volt} : U[1, 2]$ $k_b : U[1, 2]$ $T : U[1, 2]$	Feynman-A
III.15.12	$2 \cdot U \cdot (1 - \cos(k \cdot d))$	$U : U[1, 5]$ $k : U[1, 5]$ $d : U[1, 5]$	Feynman-A
III.15.14	$\frac{\left(\frac{h}{2\pi}\right)^2}{2 \cdot E_n \cdot d^2}$	$h : U[1, 5]$ $E_n : U[1, 5]$ $d : U[1, 5]$	Feynman-A
III.15.27	$\frac{2\pi \cdot \alpha}{n \cdot d}$	$\alpha : U[1, 5]$ $n : U[1, 5]$ $d : U[1, 5]$	Feynman-A
III.17.37	$\beta \cdot (1 + \alpha \cdot \cos(\theta))$	$\beta : U[1, 5]$ $\alpha : U[1, 5]$ $\theta : U[1, 5]$	Feynman-A
III.19.51	$-\frac{m \cdot q^4}{2 \cdot (4\pi \cdot \epsilon)^2 \cdot \left(\frac{h}{2\pi}\right)^2} \cdot \frac{1}{n^2}$	$m : U[1, 5]$ $q : U[1, 5]$ $\epsilon : U[1, 5]$ $h : U[1, 5]$ $n : U[1, 5]$	Feynman-A
III.21.20	$-\frac{\rho_{c_0} \cdot q \cdot A_{\text{vec}}}{m}$	$\rho_{c_0} : U[1, 5]$ $q : U[1, 5]$ $A_{\text{vec}} : U[1, 5]$ $m : U[1, 5]$	Feynman-A

Tabla A.2: Funciones del *dataset* de Feynman, con las especificaciones del espacio de entradas de los conjuntos de entrenamiento y validación. También se indica el correspondiente conjunto de funciones y terminales a utilizar (*Function-Terminal Set*).

A.2. Test de Friedman

La prueba de Friedman es un test no paramétrico que tiene como hipótesis nula (H_0) que las medianas de las distribuciones de los diferentes grupos de datos son iguales. En otras palabras, su objetivo es determinar si en un conjunto de k algoritmos, con $k \geq 2$, existen al menos dos cuyos datos de rendimiento presenten diferencias significativas en sus medianas.

El procedimiento comienza ordenando de manera ascendente los datos, preservando la identificación del grupo al que pertenecen. A cada dato se le asigna un rango, de tal manera que el valor más bajo recibe el rango 1, el segundo el rango 2, y así sucesivamente. Si se produce un empate, se utiliza el promedio de los rangos correspondientes al intervalo; por ejemplo, si los valores empatados deberían ocupar los rangos 3, 4 y 5, se les asigna el valor promedio de 4.

Luego, se calcula el promedio de los rangos para cada conjunto de datos, utilizando la fórmula $R_j = \frac{1}{n}\sum_{i=1}^{n} r_{ji}$, donde $j = 1, 2, ..., k$ representa los algoritmos, n es la cantidad de datos por grupo, y r_{ji} es el rango del i-ésimo dato del algoritmo j.

A partir de los promedios de los rangos, se calcula el estadístico de prueba Q mediante la fórmula:

$$Q = \frac{12n}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right)$$

El valor de Q sigue aproximadamente una distribución χ^2 con $k-1$ grados de libertad. Finalmente, se calcula el valor p asociado a este estadístico Q y se compara con un nivel de significancia α. Si $p \geq \alpha$, se acepta la hipótesis nula, concluyendo que no hay diferencias estadísticamente significativas entre las medianas de los algoritmos.

Por otro lado, si $p < \alpha$, se rechaza la hipótesis nula, lo que indica que existen diferencias estadísticamente significativas entre las medianas de al menos dos de los algoritmos. En este caso, se puede concluir que no todos los algoritmos presentan un rendimiento similar, y se justifica realizar pruebas adicionales, como análisis *post-hoc*, para identificar específicamente cuáles algoritmos difieren entre sí (J. Ferreira, 2022; Computing y Group, s.f.).

En este trabajo, la prueba de Friedman se implementó utilizando la función `friedmanchisquare` del paquete `scipy.stats` en Python.

A.3. Ajuste de p-valores en pruebas múltiples

Cuando se realizan varias pruebas estadísticas simultáneamente, el riesgo de cometer un error de tipo I (rechazar una hipótesis nula verdadera) aumenta. Este problema se debe a que, cuanto más pruebas realizamos, mayor es la probabilidad de que alguna de ellas arroje un resultado significativo solo por azar. Este fenómeno se conoce como *inflación de la tasa de error familiar* (*family-wise error rate*, FWER). Para mitigar este riesgo, es necesario aplicar correcciones que ajusten los p-valores obtenidos de cada prueba, de manera que el nivel de significancia general del conjunto de pruebas se mantenga controlado (Navarro, 2023; Computing y Group, s.f.).

Existen varios métodos para ajustar los p-valores en este contexto. El método más simple y conocido es la corrección de Bonferroni, que ajusta el valor de significancia α dividiéndolo por el número de pruebas realizadas (Dunn, 1961). Sin embargo, este método puede ser demasiado conservador, incrementando el

riesgo de cometer errores de tipo II (falso negativo) y reduciendo el poder estadístico del análisis. Para solucionar este problema, se han propuesto métodos alternativos, como el procedimiento de Holm, que ofrece un mejor balance entre el control de la tasa de error y el poder de la prueba (Holm, 1979).

A.4. El procedimiento *post-hoc* de Holm

El procedimiento de Holm, propuesto por S. Holm en 1979, es un método secuencial *step-down* que ajusta los p-valores de manera progresiva, comenzando por el p-valor más pequeño. La idea es rechazar las hipótesis nulas en un orden secuencial, lo que permite un mayor poder que la corrección de Bonferroni sin aumentar el riesgo de errores de tipo I.

Este método funciona de la siguiente manera:

1. Ordenar los p-valores: Se ordenan los valores p de las hipótesis en cuestión de menor a mayor: $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$, donde k es el número de comparaciones (o algoritmos).

2. Ajustar los p-valores: El p-valor de la primera hipótesis se compara con $\alpha/(k-1)$. Si $p_1 < \alpha/(k-1)$, se rechaza la primera hipótesis nula y se avanza a la segunda comparación. En este caso, el siguiente p-valor se compara con $\alpha/(k-2)$. El proceso continúa hasta que una de las hipótesis no pueda ser rechazada, momento en el cual todas las hipótesis restantes se conservan.

   El p-valor ajustado para cada hipótesis H_i se calcula de la siguiente manera:

   $$\text{Holm } APV_i = \text{mín}\{v, 1\}, \quad \text{donde } v = \text{máx}\{(k-j)p_j : 1 \leq j \leq i\}$$

   donde p_j es el p-valor obtenido para la hipótesis j, y k es el número total de comparaciones.

3. Interpretación: Si en algún punto no se puede rechazar una hipótesis, las hipótesis restantes se retienen, ya que se considera que no existe evidencia suficiente para demostrar diferencias estadísticamente significativas entre los grupos.

El procedimiento de Holm es más poderoso que la corrección de Bonferroni, ya que permite mantener el mismo nivel de control sobre los errores de tipo I, pero con una menor tasa de errores de tipo II (Navarro, 2023).

En este trabajo, el procedimiento de corrección de Holm se implementó utilizando la función `posthoc_siegel_friedman(data, p_adjust = holm)` del paquete `scikit_posthocs` en Python.

A.5. Resultados de la experimentación siguiendo un enfoque monoobjetivo

En la Tabla A.3, se utiliza el símbolo ✓ en la columna *éxito* para indicar cuando un algoritmo logró obtener, en al menos una de sus 30 ejecuciones independientes, una expresión matemática equivalente a la original. También se considera un éxito cuando el algoritmo logra un error para el conjunto de entrenamiento igual a cero o muy cercano a este valor, aunque dicha expresión no sea exactamente equivalente a la buscada. En caso contrario, se marca con el símbolo × en esa columna. A partir de aquí, cuando se refiera a la recuperación de la expresión matemática original, no se descarta la posibilidad de que el algoritmo haya llegado a una expresión diferente y simplemente se haya detenido al alcanzar la recompensa establecida. Sin embargo, por simplicidad, esto se denominará como la recuperación de la expresión o función original.

En la Tabla A.4, el símbolo ✓ indica la existencia de diferencias estadísticamente significativas entre las distribuciones de RMSE de los algoritmos, de acuerdo con el Test de Friedman. Por otro lado, el símbolo × se utiliza cuando no se encuentran diferencias estadísticamente significativas en la columna *Friedman Test Statistic*.

Función	**DSO**						**Operon**						**DSR**					
	Med	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}
Vladislavleva																		
1	**1.38e-02**	1.38e-02	7.47e-02	3.27e-01	×	1.74e+03	1.56e-01	1.37e-01	6.51e-02	5.67e+01	×	4.18e-01	1.25e-01	2.57e-02	1.14e-01	3.35e-01	×	1.04e+02
2	**4.50e-02**	4.50e-02	1.64e-01	2.21e+00	×	1.34e+03	1.48e-01	8.82e-02	4.41e-02	2.25e-01	×	4.56e-01	2.16e-01	5.66e-02	1.11e-01	8.71e-01	×	1.30e+02
3	**1.54e-01**	1.54e-01	8.82e-01	1.21e+00	×	1.19e+03	7.36e-01	5.06e-01	2.63e-01	1.68e+00	×	6.73e-01	1.15e+00	2.48e-01	7.99e-01	1.31e+00	×	2.08e+03
4	**1.47e-02**	1.47e-02	1.86e-01	9.29e-01	×	1.72e+03	1.77e-01	7.86e-03	1.56e-01	3.08e+01	×	7.13e-01	2.01e-01	1.00e-02	1.86e-01	1.96e+00	×	1.02e+02
5	**1.56e-01**	1.56e-01	3.85e-01	3.93e+00	×	2.03e+03	4.10e-01	3.49e-01	3.02e-02	6.16e-01	×	4.60e-01	6.64e-01	1.22e-01	4.24e-01	7.60e-01	×	1.46e+02
6	2.84e+01	2.84e+01	2.65e+00	1.07e+02	×	2.04e+03	5.21e+00	6.73e+00	2.70e+00	1.15e+02	×	4.43e-01	1.40e+01	1.88e+01	3.56e+00	9.01e+01	×	1.99e+02
7	2.31e-01	2.31e-01	3.79e+00	1.01e+01	×	1.56e+03	1.94e+00	1.33e+00	9.22e-01	3.06e+00	×	6.06e-01	4.16e+00	3.17e-01	3.79e+00	4.77e+00	×	1.90e+02
8	6.06e+00	6.06e+00	2.03e+00	7.11e+01	×	2.03e+03	1.68e+00	7.15e-02	1.46e+00	1.33e+01	×	3.89e-01	2.19e+00	8.74e+00	1.72e+00	1.71e+02	×	1.32e+02
Feynman																		
I.6.2a	3.35e-03	1.87e-03	1.27e-03	6.22e-03	×	1.65e+03	**4.48e-04**	4.68e-04	1.17e-04	1.11e-03	×	5.74e+00	5.74e+00	1.76e-03	**2.12e-09**	5.51e-03	✓	2.03e+02
I.6.2	2.01e-02	2.75e-03	1.09e-02	2.16e-02	×	1.88e+03	**6.98e-03**	3.46e-03	**3.66e-03**	1.14e-02	×	5.25e+00	5.25e+00	4.17e-04	1.77e-02	2.15e-02	×	1.91e+02
I.6.2b	3.42e-02	2.10e-04	3.24e-02	3.46e-02	×	1.68e+03	**2.05e-02**	1.32e-02	**8.96e-03**	3.38e-02	×	6.12e+00	6.12e+00	9.03e-05	3.40e-02	3.47e-02	×	1.77e+02
I.8.14	8.97e-01	4.53e-02	7.42e-01	9.45e-01	×	1.63e+03	**7.14e-01**	9.21e-03	**4.72e-01**	7.65e-01	×	5.01e+00	5.01e+00	5.58e-02	8.03e-01	9.59e-01	×	1.96e+02
I.9.18	8.76e-02	7.07e-03	6.54e-02	9.54e-02	×	1.75e+03	**5.56e-02**	5.49e-03	**4.61e-02**	6.58e-02	×	7.25e+00	7.25e+00	1.01e-02	5.65e-02	9.60e-02	×	2.36e+02
I.10.7	7.31e-02	3.99e-02	1.78e-02	9.49e-02	×	1.63e+03	**4.08e-02**	2.10e-02	**1.37e-02**	7.59e-02	×	4.44e+00	4.44e+00	1.97e-02	2.06e-02	1.15e-01	×	1.56e+02
I.11.19	4.87e+00	5.13e-01	3.96e+00	6.18e+00	×	5.47e+03	**2.36e+00**	6.83e-01	**5.53e-04**	5.06e+00	✓	5.74e+00	5.74e+00	3.88e-01	3.41e+00	4.58e+00	×	2.16e+02
I.12.1	**8.17e-16**	9.85e-16	**0.00e+00**	3.80e-06	✓	3.61e+01	1.35e-05	1.49e-08	1.35e-05	1.35e-05	✓	2.82e+00	2.82e+00	6.79e-10	**0.00e+00**	3.80e-06	✓	8.00e+00
I.12.2	4.00e-02	4.51e-03	1.63e-02	4.40e-02	×	1.40e+03	**2.93e-06**	8.10e-06	**2.93e-06**	5.63e-02	×	4.27e+00	4.27e+00	1.36e-03	1.73e-02	4.55e-02	×	2.05e+02
I.12.4	6.22e-03	4.63e-03	1.39e-03	1.06e-02	×	1.76e+03	**2.88e-06**	1.01e-10	2.87e-06	2.88e-06	×	3.92e+00	3.92e+00	1.93e-03	**9.18e-10**	1.19e-02	✓	2.58e+02

Función	DSO						Operon						DSR					
	Med	IQR	Min	Max	Éxito	t_{avg}	Med	IQR	Min	Max	Éxito	t_{avg}	Med	IQR	Min	Max	Éxito	t_{avg}
I.12.5	**8.24e-16**	8.65e-09	**0.00e+00**	7.92e-07	✓	7.61e+01	1.35e-05	1.51e-08	1.35e-05	1.36e-05	✓	2.74e+00	2.74e+00	8.65e-09	**0.00e+00**	7.92e-07	✓	8.44e+00
I.12.11	1.39e+01	2.27e+00	5.22e+00	1.67e+01	×	1.68e+03	**5.00e+00**	5.20e-01	**4.38e+00**	1.55e+01	×	8.60e+00	8.60e+00	3.30e+00	9.22e+00	1.64e+01	×	2.63e+02
I.13.4	7.51e+00	1.64e+00	4.13e+00	9.58e+00	×	1.79e+03	4.85e+00	8.53e-01	**3.13e+00**	7.58e+00	×	4.13e+00	**4.13e+00**	2.83e+00	5.92e+00	1.09e+01	×	1.78e+03
I.13.12	5.24e+00	4.79e-01	4.19e+00	5.87e+00	×	1.84e+03	**4.29e+00**	4.33e+00	**4.27e-05**	5.54e+00	×	9.27e+00	9.27e+00	3.47e-01	4.25e+00	5.76e+00	×	2.80e+02
I.14.3	**2.94e-15**	2.33e-15	**0.00e+00**	5.17e-07	✓	2.58e+02	5.20e-05	6.68e-08	5.20e-05	5.22e-05	✓	3.85e+00	3.85e+00	1.35e-15	**0.00e+00**	1.32e-05	✓	3.83e+01
I.14.4	**1.25e-07**	4.31e-01	1.06e-17	1.18e+00	✓	1.46e+03	3.49e-05	5.09e-08	3.49e-05	3.50e-05	✓	5.15e+00	5.15e+00	7.53e-16	**0.00e+00**	2.29e-15	✓	9.55e+01
I.15.3x	2.03e-01	4.69e-06	1.76e-01	2.49e-01	×	2.40e+03	**1.20e-01**	2.11e-02	**2.19e-02**	2.16e-01	×	1.01e+01	1.01e+01	3.05e-16	1.78e-01	3.05e-01	×	1.60e+02
I.15.3t	3.14e+00	3.95e-01	2.57e+00	3.58e+00	×	2.01e+03	**1.53e+00**	4.01e-01	**1.22e+00**	3.16e+00	×	4.70e+00	4.70e+00	4.07e-01	2.16e+00	3.43e+00	×	1.63e+02
I.15.1	2.10e-01	5.35e-02	4.60e-02	2.54e-01	×	1.70e+03	**9.62e-02**	4.32e-02	**1.13e-02**	1.55e-01	×	5.88e+00	5.88e+00	1.97e-02	1.82e-01	2.79e-01	×	1.92e+02
I.16.6	3.41e-01	3.00e-02	2.52e-01	3.63e-01	×	1.60e+03	**2.54e-01**	8.71e-02	**1.25e-01**	3.61e-01	×	6.29e+00	6.29e+00	2.12e-02	2.96e-01	3.76e-01	×	1.58e+02
I.18.4	2.56e-01	4.56e-02	2.50e-01	3.07e-01	×	1.67e+03	**2.07e-01**	1.08e-02	**8.90e-02**	2.31e-01	×	5.37e+00	5.37e+00	0.00e+00	2.50e-01	2.98e-01	×	2.08e+02
I.18.12	**8.48e-16**	3.76e-01	**0.00e+00**	2.27e+00	✓	1.27e+03	8.26e-03	3.51e-02	2.25e-05	3.70e+00	✓	6.06e+00	6.06e+00	2.06e-07	**0.00e+00**	9.10e-07	✓	9.97e+01
I.18.14	9.22e+00	7.07e-01	2.55e-15	1.21e+01	✓	1.78e+03	**3.72e-02**	3.65e-01	9.04e-04	1.34e+01	✓	6.48e+00	6.48e+00	9.22e+00	**2.32e-15**	1.07e+01	✓	2.41e+02
I.24.6	4.01e+00	1.15e+00	2.24e+00	5.60e+00	×	1.81e+03	**1.88e+00**	8.92e-01	**1.05e+00**	3.26e+00	×	6.05e+00	6.05e+00	7.36e-01	2.51e+00	4.75e+00	×	2.36e+02
I.25.13	**1.34e-16**	2.68e-09	**0.00e+00**	2.55e-07	✓	5.76e+01	4.22e-06	5.28e-09	4.21e-06	4.23e-06	✓	4.68e+00	4.68e+00	2.74e-09	**0.00e+00**	4.91e-07	✓	9.82e+00
I.26.2	**4.43e-02**	1.19e-02	**3.13e-02**	7.33e-02	×	1.71e+03	4.19e-01	8.63e-02	2.54e-01	5.67e-01	×	7.12e+00	7.12e+00	3.50e-03	**3.13e-02**	5.24e-02	×	2.90e+02
I.27.6	1.12e-01	3.36e-02	**6.12e-17**	1.46e-01	✓	1.56e+03	**4.70e-02**	3.06e-02	1.68e-02	1.03e-01	×	6.24e+00	6.24e+00	3.77e-02	9.01e-17	1.48e-01	✓	2.50e+02
I.29.4	**1.79e-16**	5.77e-10	**0.00e+00**	1.21e-06	✓	4.48e+01	4.56e-06	3.54e-09	4.56e-06	4.58e-06	✓	5.22e+00	5.22e+00	2.76e-12	**0.00e+00**	1.21e-06	✓	7.50e+00
I.29.16	1.60e+00	4.64e-02	1.45e+00	1.73e+00	×	1.80e+03	**1.16e+00**	2.97e-01	**6.79e-01**	1.67e+00	×	6.53e+00	6.53e+00	1.12e-01	1.43e+00	1.73e+00	×	2.75e+02
I.30.3	1.94e+00	7.50e-02	1.70e+00	1.99e+00	×	1.74e+03	**1.74e-07**	4.21e-08	**1.22e-07**	2.45e-07	×	3.95e+00	3.95e+00	5.12e-02	1.88e+00	2.03e+00	×	3.04e+02
I.30.5	**9.66e-03**	1.89e-03	3.93e-03	9.95e-03	×	1.61e+03	1.21e-02	4.36e-03	**2.61e-03**	2.05e-02	×	6.80e+00	6.80e+00	1.31e-03	4.00e-03	9.95e-03	×	1.92e+02
I.32.5	4.30e-01	8.95e-02	2.59e-01	5.27e-01	×	1.81e+03	**1.62e-01**	2.69e-02	**4.89e-02**	2.54e-01	×	6.08e+00	6.08e+00	4.92e-02	2.59e-01	5.00e-01	×	3.01e+02
I.32.17	3.11e+00	3.03e-01	2.10e+00	3.31e+00	×	2.07e+03	**2.09e+00**	8.31e-01	**1.31e+00**	3.13e+00	×	9.70e+00	9.70e+00	2.73e-01	2.50e+00	3.26e+00	×	3.66e+02
I.34.8	**1.87e-15**	3.23e-16	**0.00e+00**	1.78e+00	✓	1.04e+03	3.63e-05	1.15e-07	3.61e-05	3.10e+00	✓	5.17e+00	5.17e+00	1.73e-16	1.18e-15	2.89e+00	✓	1.28e+02
I.34.1	**3.54e+00**	6.11e+00	**1.02e-14**	9.10e+00	✓	1.52e+03	1.46e+01	1.66e+01	5.81e-04	1.80e+01	✓	7.80e+00	7.80e+00	5.54e+00	**1.02e-14**	9.43e+00	✓	2.84e+02
I.34.14	4.39e-02	9.95e-03	1.42e-02	5.95e-02	×	1.59e+03	**1.09e-02**	7.97e-03	**7.41e-03**	3.85e-02	×	6.98e+00	6.98e+00	1.55e-02	1.42e-02	6.71e-02	×	2.45e+02
I.34.27	**5.58e-08**	1.59e-02	**4.56e-08**	1.53e-01	✓	1.15e+03	3.61e-06	1.64e-09	3.61e-06	3.61e-06	✓	4.32e+00	4.32e+00	3.30e-18	**4.56e-08**	8.94e-02	✓	1.03e+02
I.37.4	1.44e+00	1.88e-01	1.00e+00	1.58e+00	×	1.81e+03	**9.93e-01**	3.94e-01	**1.83e-01**	1.17e+00	×	6.90e+00	6.90e+00	1.75e-01	9.51e-01	1.60e+00	×	2.89e+02
I.38.12	6.33e-01	1.59e-01	3.02e-01	8.50e-01	×	1.79e+03	**9.03e-06**	1.66e-08	**9.00e-06**	3.09e-01	✓	5.75e+00	5.75e+00	1.21e-01	1.59e-01	8.31e-01	×	2.78e+02
I.39.1	3.33e-01	2.94e-01	**0.00e+00**	5.70e-01	✓	1.59e+03	**1.98e-05**	1.37e-08	1.97e-05	1.98e-05	✓	4.74e+00	4.74e+00	3.29e-01	**0.00e+00**	7.32e-01	✓	1.60e+02
I.39.11	6.60e-01	6.69e-01	**4.32e-16**	9.55e-01	✓	1.63e+03	**1.12e-02**	2.68e-02	1.24e-05	6.38e-02	✓	7.58e+00	7.58e+00	2.69e-01	5.95e-16	9.93e-01	✓	2.74e+02
I.39.22	**1.86e-15**	3.57e-16	1.43e-15	2.47e+00	✓	9.87e+02	3.64e-05	9.88e-08	3.63e-05	2.16e+00	✓	5.28e+00	5.28e+00	1.66e-16	**1.19e-15**	3.35e+00	✓	1.32e+02
I.40.1	4.22e-01	4.40e-02	3.28e-01	4.58e-01	×	1.90e+03	**2.72e-01**	1.03e-01	**2.39e-01**	4.41e-01	×	1.32e+01	1.32e+01	2.60e-02	3.39e-01	4.66e-01	×	3.51e+02
I.41.16	1.52e+00	1.42e-01	8.43e-01	1.67e+00	×	2.00e+03	**1.80e-01**	5.85e-02	**1.33e-01**	4.36e-01	×	1.68e+01	1.68e+01	4.90e-01	8.43e-01	1.53e+00	×	3.67e+02
I.43.16	**1.87e-15**	8.98e-01	1.45e-15	3.56e+00	✓	1.08e+03	3.52e-05	7.05e-08	3.51e-05	2.79e+00	×	8.37e+00	8.37e+00	5.15e-16	**0.00e+00**	4.25e+00	✓	1.12e+02
I.43.31	**3.13e-15**	2.71e-15	**0.00e+00**	8.92e-07	✓	2.36e+02	5.17e-05	4.47e-08	5.16e-05	5.18e-05	×	8.21e+00	8.21e+00	1.16e-15	**0.00e+00**	1.09e-05	✓	3.60e+01
I.43.43	6.25e-01	1.81e-01	**3.40e-16**	8.69e-01	✓	1.72e+03	**2.70e-02**	3.67e-02	7.47e-06	2.18e-01	✓	8.29e+00	8.29e+00	1.84e-01	3.44e-01	7.06e-01	×	2.73e+02
I.44.4	**8.21e+00**	2.03e+00	8.04e+00	1.06e+01	×	2.07e+03	1.05e+01	6.99e+00	**6.78e-05**	1.13e+01	✓	1.45e+01	1.45e+01	2.03e+00	6.14e+00	1.10e+01	×	4.30e+02
I.47.23	**1.82e-16**	3.83e-09	**0.00e+00**	2.13e-01	✓	9.24e+02	6.14e-02	4.85e-02	3.59e-06	1.48e-01	✓	7.89e+00	7.89e+00	8.63e-17	**0.00e+00**	1.96e-01	✓	1.10e+02
I.48.2	9.20e-01	0.00e+00	9.20e-01	9.20e-01	×	1.60e+03	**1.82e-01**	6.56e-02	**9.00e-02**	2.50e-01	×	5.62e+00	5.62e+00	1.11e-01	3.42e-01	6.15e-01	×	8.55e+01

Función	**DSO**						**Operon**						**DSR**					
	Med	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}
I.50.26	**1.54e+00**	1.32e-01	1.36e+00	1.69e+00	×	2.02e+03	1.57e+00	2.02e-01	**9.55e-01**	1.61e+00	×	6.36e+00	6.36e+00	1.47e-01	1.35e+00	1.67e+00	×	2.96e+02
II.2.42	4.24e+00	1.39e-01	3.21e+00	4.71e+00	×	2.34e+03	**6.72e-03**	2.61e+00	**2.62e-05**	4.56e+00	✓	9.49e+00	9.49e+00	3.82e-01	2.76e+00	4.31e+00	×	3.84e+02
II.3.24	4.35e-03	4.47e-03	**2.06e-09**	1.31e-02	✓	1.95e+03	**2.90e-06**	1.58e-09	2.90e-06	5.64e-04	✓	6.06e+00	6.06e+00	1.54e-19	**2.06e-09**	1.02e-02	✓	1.93e+02
II.4.23	7.15e-03	7.00e-03	8.51e-04	1.48e-02	×	1.87e+03	**2.89e-06**	1.66e-09	2.89e-06	1.31e-02	✓	6.26e+00	6.26e+00	2.60e-03	**1.43e-09**	6.85e-03	✓	2.24e+02
II.6.11	1.27e-02	2.26e-03	7.27e-03	1.59e-02	×	1.88e+03	**2.56e-03**	5.30e-03	**1.11e-05**	1.07e-02	✓	7.84e+00	7.84e+00	2.86e-03	7.99e-03	1.47e-02	×	2.74e+02
II.6.15a	1.93e-01	1.81e-02	1.53e-01	2.16e-01	×	2.74e+03	**9.32e-02**	3.89e-02	**6.76e-02**	2.24e-01	×	1.21e+01	1.21e+01	4.24e-02	9.58e-02	1.48e-01	×	3.40e+02
II.6.15b	2.01e-02	2.98e-03	1.29e-02	2.48e-02	×	2.09e+03	**9.09e-03**	6.35e-03	**3.09e-05**	2.01e-02	✓	8.74e+00	8.74e+00	2.27e-03	1.84e-02	2.62e-02	×	2.60e+02
II.8.7	3.07e-02	5.39e-03	1.12e-02	4.47e-02	×	1.97e+03	**2.91e-06**	2.11e-08	**2.91e-06**	5.02e-03	✓	6.95e+00	6.95e+00	2.03e-03	1.08e-02	3.34e-02	×	2.20e+02
II.8.31	**2.07e-15**	3.75e-01	**0.00e+00**	1.25e+00	✓	1.40e+03	3.48e-05	4.96e-08	3.47e-05	3.49e-05	✓	5.08e+00	5.08e+00	1.08e-15	**0.00e+00**	4.80e-01	✓	7.69e+01
II.10.9	2.76e-02	4.71e-02	**0.00e+00**	7.54e-02	✓	1.46e+03	**3.86e-05**	2.00e-03	3.03e-06	1.65e-02	✓	5.88e+00	5.88e+00	2.36e-02	3.72e-17	8.52e-02	✓	1.87e+02
II.11.3	5.66e-02	1.56e-02	2.45e-02	7.30e-02	×	2.09e+03	**2.20e-02**	2.49e-02	**2.54e-03**	5.50e-02	✓	9.89e+00	9.89e+00	1.18e-02	3.39e-02	7.03e-02	×	3.28e+02
II.11.17	9.82e-01	1.64e-02	8.89e-01	1.01e+00	×	2.37e+03	**5.74e-01**	2.58e-01	**4.57e-01**	9.12e-01	×	9.94e+00	9.94e+00	1.43e-02	9.17e-01	1.02e+00	×	3.92e+02
II.11.20	3.82e+00	1.04e+00	2.51e+00	4.36e+00	×	2.05e+03	**2.80e-05**	8.91e-08	**2.79e-05**	1.11e+00	✓	9.56e+00	9.56e+00	9.05e-01	2.61e+00	4.62e+00	×	3.92e+02
II.11.27	1.14e-01	5.71e-02	6.12e-02	1.88e-01	×	2.89e+03	**4.21e-02**	2.33e-02	**6.79e-03**	6.83e-02	×	6.19e+00	6.19e+00	2.22e-02	2.99e-02	6.73e-02	×	2.28e+02
II.11.28	1.53e-02	2.09e-03	8.23e-03	1.72e-02	×	1.59e+03	**2.80e-03**	3.99e-04	**1.20e-05**	8.54e-03	✓	7.24e+00	7.24e+00	1.47e-08	1.04e-02	1.53e-02	×	2.02e+02
II.13.17	1.07e-02	1.70e-03	4.43e-03	1.39e-02	×	1.81e+03	**2.88e-06**	2.21e-08	**2.88e-06**	1.22e-02	✓	7.91e+00	7.91e+00	3.69e-03	1.81e-04	1.31e-02	✓	2.55e+02
II.13.23	6.25e-02	5.77e-02	1.92e-02	9.69e-02	×	2.26e+03	**4.08e-02**	1.75e-02	**2.91e-03**	8.97e-02	×	7.36e+00	7.36e+00	3.21e-02	1.92e-02	1.16e-01	×	1.83e+02
II.13.34	1.99e-01	1.56e-02	1.37e-01	2.30e-01	×	1.82e+03	**9.71e-02**	3.68e-02	**2.95e-02**	1.49e-01	×	7.24e+00	7.24e+00	1.68e-02	1.51e-01	2.44e-01	×	1.63e+02
II.15.4	3.23e-01	1.67e+00	**0.00e+00**	2.33e+00	✓	1.82e+03	**2.48e-02**	3.49e-01	8.45e-05	3.20e+00	✓	8.04e+00	8.04e+00	3.42e-01	**0.00e+00**	1.85e+00	✓	1.71e+02
I.15.5	**3.47e-07**	6.50e-01	**0.00e+00**	2.20e+00	✓	1.24e+03	1.48e-02	4.69e-01	1.94e-04	3.25e+00	✓	6.03e+00	6.03e+00	1.42e+00	**0.00e+00**	2.39e+00	✓	2.55e+02
II.21.32	1.60e-02	6.04e-03	9.18e-03	2.72e-02	×	2.15e+03	**7.35e-03**	2.52e-03	**2.94e-03**	2.24e-02	×	8.11e+00	8.11e+00	2.27e-03	8.07e-03	2.87e-02	×	3.29e+02
II.24.17	9.13e-02	1.82e-02	2.05e-02	1.11e-01	×	1.97e+03	**3.17e-02**	1.56e-02	**1.24e-02**	8.62e-02	×	5.14e+00	5.14e+00	4.77e-03	6.65e-02	1.01e-01	×	2.24e+02
II.27.16	**5.72e-15**	1.03e-15	**0.00e+00**	1.27e+00	✓	9.18e+02	1.40e-04	2.02e-07	1.40e-04	8.16e+00	✓	3.57e+00	3.57e+00	9.62e-17	**0.00e+00**	2.68e+00	✓	7.70e+01
II.27.18	**3.64e-15**	2.07e-16	**0.00e+00**	4.65e-15	✓	1.60e+02	6.74e-05	6.77e-08	6.73e-05	6.75e-05	✓	3.09e+00	3.09e+00	8.90e-16	**0.00e+00**	1.07e-06	✓	1.98e+01
II.34.2a	8.58e-02	3.65e-02	**2.06e-08**	1.39e-01	✓	2.04e+03	**3.35e-06**	1.57e-09	3.35e-06	3.37e-06	✓	3.61e+00	3.61e+00	5.25e-02	**2.06e-08**	1.68e-01	✓	2.42e+02
II.34.2	9.70e-01	1.76e+00	**0.00e+00**	2.03e+00	✓	1.76e+03	**2.63e-05**	3.81e-08	2.62e-05	2.63e-05	✓	3.37e+00	3.37e+00	5.51e-01	2.15e-18	1.54e+00	✓	1.83e+02
II.34.11	1.66e+00	4.94e-01	9.63e-01	2.54e+00	×	2.05e+03	**1.86e-05**	2.56e-08	**1.86e-05**	1.45e+00	✓	4.03e+00	4.03e+00	4.59e-02	1.09e+00	2.27e+00	×	2.82e+02
II.34.29a	7.20e-02	2.62e-02	2.99e-02	1.15e-01	×	2.07e+03	**2.99e-06**	1.85e-09	**2.99e-06**	5.00e-02	✓	5.44e+00	5.44e+00	3.80e-02	1.24e-02	1.33e-01	×	2.59e+02
II.34.29b	2.00e+01	1.20e+00	1.52e+01	2.19e+01	×	2.40e+03	**7.46e+00**	1.16e+01	**1.87e-04**	1.71e+01	✓	7.62e+00	7.62e+00	1.16e+00	1.54e+01	1.95e+01	×	2.76e+02
I.35.18	1.86e-01	1.96e-02	1.65e-01	2.08e-01	×	2.21e+03	**9.46e-02**	1.14e-02	**6.32e-02**	1.24e-01	×	6.43e+00	6.43e+00	2.06e-02	1.67e-01	2.16e-01	×	3.35e+02
II.35.21	**2.90e-01**	2.70e-02	**1.82e-01**	3.26e-01	×	3.05e+03	1.14e+00	1.47e-01	8.31e-01	1.52e+00	×	1.24e+01	1.24e+01	1.40e-01	1.76e+00	2.12e+00	×	3.07e+02
II.36.38	6.61e-01	8.18e-02	4.94e-01	7.49e-01	×	4.33e+03	**3.64e-01**	1.80e-01	**2.53e-01**	6.17e-01	×	2.85e+01	2.85e+01	7.91e-02	4.89e-01	7.05e-01	×	2.22e+02
II.37.1	**4.53e-15**	6.55e-07	**0.00e+00**	8.79e-01	✓	1.46e+03	5.27e-04	1.16e-03	6.01e-05	1.31e+00	✓	4.23e+00	4.23e+00	8.19e-16	3.71e-15	4.77e-15	✓	1.23e+02
II.38.3	**1.86e-15**	1.15e+00	1.29e-15	3.22e+00	✓	1.34e+03	3.60e-05	4.19e-08	3.59e-05	2.17e+00	✓	3.61e+00	3.61e+00	2.28e-16	**0.00e+00**	3.22e+00	✓	1.22e+02
II.38.14	2.51e-02	1.48e-02	5.35e-03	4.47e-02	×	1.94e+03	**6.22e-05**	3.68e-03	2.97e-06	7.71e-03	✓	4.03e+00	4.03e+00	1.08e-02	**0.00e+00**	3.52e-02	✓	2.16e+02
III.4.32	2.44e-01	3.63e-02	1.72e-01	3.05e-01	×	2.15e+03	**1.41e-01**	5.58e-02	**9.11e-02**	2.69e-01	×	4.58e+00	4.58e+00	4.33e-02	1.97e-01	3.03e-01	×	2.65e+02
III.4.33	3.67e-01	4.84e-02	2.87e-01	4.37e-01	×	1.92e+03	**1.54e-01**	8.78e-02	**3.27e-02**	2.90e-01	×	3.73e+00	3.73e+00	1.77e-02	3.25e-01	4.04e-01	×	2.08e+02
III.7.38	5.36e+00	3.94e+00	1.53e+00	9.57e+00	×	2.33e+03	**8.80e-05**	8.31e-08	**8.79e-05**	7.49e+00	✓	3.53e+00	3.53e+00	2.76e+00	1.29e+00	9.65e+00	×	1.87e+02
III.8.54	5.18e-01	1.14e-01	**1.45e-07**	5.97e-01	✓	2.22e+03	**4.93e-01**	6.98e-02	1.99e-03	6.02e-01	×	4.89e+00	4.89e+00	9.00e-02	3.16e-01	6.28e-01	×	2.05e+02
II.9.52	1.21e+01	1.47e-01	1.06e+01	1.25e+01	×	2.44e+03	1.12e+01	2.11e+00	**6.00e+00**	1.17e+01	×	7.10e+00	**7.10e+00**	1.65e-01	1.18e+01	1.22e+01	×	2.43e+02

Función	DSO						Operon						DSR					
	Med	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}	**Med**	**IQR**	**Min**	**Max**	**Éxito**	t_{avg}
III.10.19	2.20e+00	4.01e-01	1.75e+00	2.64e+00	×	2.24e+03	**1.18e+00**	8.13e-01	**5.85e-01**	1.53e+00	×	3.92e+00	3.92e+00	1.15e-01	1.60e+00	2.48e+00	×	1.50e+02
III.12.43	**5.58e-08**	1.65e-02	**4.55e-08**	1.47e-01	✓	1.28e+03	3.57e-06	1.57e-09	3.57e-06	3.57e-06	✓	3.12e+00	3.12e+00	1.54e-08	**4.55e-08**	7.10e-02	✓	6.95e+01
III.13.18	1.79e+01	1.37e+00	1.36e+01	2.05e+01	×	1.90e+03	7.46e+00	9.72e+00	**2.65e-04**	1.24e+01	✓	4.12e+00	**4.12e+00**	5.31e-01	1.07e+01	1.87e+01	×	1.66e+02
III.14.14	4.13e+00	4.50e-01	2.99e+00	4.53e+00	×	2.39e+03	**2.25e+00**	5.05e-01	**1.14e+00**	4.34e+00	×	7.05e+00	7.05e+00	2.95e-01	3.03e+00	4.46e+00	×	2.62e+02
III.15.12	**4.17e+00**	5.52e-01	2.27e+00	4.49e+00	×	1.99e+03	4.29e+00	1.38e+00	**6.47e-02**	4.44e+00	✓	4.92e+00	4.92e+00	1.46e-01	2.59e+00	4.49e+00	×	3.09e+02
III.15.14	6.19e-03	1.81e-03	4.96e-03	9.52e-03	×	1.99e+03	**2.88e-06**	5.23e-10	**2.88e-06**	2.39e-03	✓	4.63e+00	4.63e+00	2.00e-03	1.45e-03	9.58e-03	×	2.33e+02
III.15.27	7.19e-01	2.72e-01	**1.12e-07**	9.83e-01	✓	1.77e+03	**1.16e-05**	1.28e-07	1.15e-05	1.17e-05	✓	4.32e+00	4.32e+00	4.73e-01	**1.12e-07**	1.29e+00	✓	2.67e+02
III.17.37	1.62e+00	8.17e-01	**4.71e-16**	2.68e+00	✓	2.00e+03	**8.56e-01**	7.13e-02	1.85e-01	2.86e+00	×	4.80e+00	4.80e+00	1.51e+00	**4.71e-16**	1.85e+00	✓	3.54e+02
III.19.51	1.75e+00	2.34e-01	1.04e+00	1.87e+00	×	2.22e+03	**1.15e+00**	1.42e-01	**3.10e-01**	1.23e+00	×	6.90e+00	6.90e+00	9.74e-02	9.99e-01	1.89e+00	×	3.83e+02
III.21.20	**2.04e-07**	2.99e+00	1.38e-15	4.55e+00	✓	1.47e+03	3.69e-05	5.17e-08	3.69e-05	3.71e-05	✓	3.30e+00	3.30e+00	3.83e+00	**0.00e+00**	4.98e+00	✓	1.95e+02
Func. IPQ																		
Esq1. CP1	5.71e-03	7.78e-03	3.29e-04	1.24e-02	×	1.94e+03	2.21e-02	3.90e-02	**4.22e-06**	4.76e-02	×	3.23e-01	**2.01e-03**	9.17e-04	1.09e-03	5.88e-03	×	1.11e+02
Esq1. CP2	**3.47e-02**	1.06e-02	1.13e-02	4.26e-02	×	1.66e+03	3.51e-02	3.20e-01	9.51e-03	3.88e-01	×	3.77e-01	**3.47e-02**	1.77e-02	**2.21e-03**	4.26e-02	×	8.41e+01
Esq1. CP3	**3.25e-02**	2.17e-02	1.69e-02	5.63e-02	×	1.92e+03	5.28e-01	4.05e-01	2.07e-01	1.17e+00	×	4.28e-01	5.27e-02	2.31e-02	**1.66e-02**	7.54e-02	×	8.80e+01
Esq1. CP4	**8.73e-02**	1.18e-02	**5.58e-02**	7.62e+02	×	1.72e+03	5.71e-01	2.53e-01	1.44e-01	1.88e+00	×	4.36e-01	9.65e-02	1.28e-02	7.69e-02	1.11e-01	×	1.10e+02
Esq2. CP1	6.83e-02	3.08e-02	3.41e-02	1.20e-01	✓	1.65e+03	**1.19e-05**	1.15e-04	**3.01e-06**	5.04e-04	✓	4.15e-01	6.45e-02	2.58e-02	3.40e-02	9.94e-02	✓	1.15e+02
Esq2. CP2	3.37e-02	0.00e+00	1.94e-02	5.28e-02	✓	1.74e+03	**8.95e-05**	2.28e-04	**3.08e-06**	8.93e-04	✓	4.98e-01	3.37e-02	0.00e+00	1.93e-02	4.84e-02	✓	1.14e+02
Esq2. CP3	8.03e+00	7.44e-01	6.34e+00	8.56e+00	✓	2.64e+03	**1.34e-04**	3.02e-04	**3.35e-06**	1.05e-03	×	4.16e-01	8.03e+00	3.74e-03	5.48e+00	9.99e+00	×	1.02e+02
Esq2. CP4	8.62e+00	2.49e+00	6.84e+00	1.87e+01	×	1.69e+03	**3.27e-02**	7.18e-03	**2.02e-02**	4.23e-02	×	8.38e-01	1.31e+01	7.78e+00	7.08e+00	2.62e+01	×	1.51e+02
Esq3. CP1	1.10e-03	4.32e-03	3.32e-04	1.73e-02	✓	2.16e+03	**1.07e-05**	8.81e-05	**2.97e-06**	3.20e-04	✓	4.66e-01	2.03e-03	4.01e-03	1.10e-03	5.89e-03	✓	8.68e+01
Esq3. CP2	3.36e-02	1.17e-02	2.26e-03	3.82e-02	✓	1.90e+03	**6.02e-05**	1.76e-04	**2.87e-06**	9.36e-04	×	4.98e-01	3.55e-02	2.05e-02	9.12e-03	3.82e-02	✓	7.15e+01
Esq3. CP3	4.04e-02	1.92e-02	2.80e-02	5.50e-02	×	3.94e+03	**1.04e-02**	7.27e-03	**6.52e-03**	2.47e-02	×	4.36e-01	4.08e-02	1.75e-02	2.80e-02	5.48e-02	×	6.03e+01
Esq3. CP4	8.81e-03	4.04e-03	8.27e-03	1.45e-02	×	2.22e+03	**1.91e-03**	7.03e-04	**1.19e-03**	3.18e-03	×	7.65e-01	8.69e-03	4.23e-04	8.27e-03	1.34e-02	×	1.23e+02
Esq4. CP1	1.04e-03	5.83e-03	3.12e-04	1.18e-02	✓	1.87e+03	**2.31e-05**	1.10e-04	**3.05e-06**	3.86e-04	✓	3.83e-01	1.91e-03	3.70e-03	1.04e-03	6.86e-03	✓	1.07e+02
Esq4. CP2	3.50e-02	4.62e-03	2.23e-03	4.86e-02	×	1.72e+03	**9.23e-05**	3.39e-04	**3.03e-06**	7.64e-04	✓	4.72e-01	3.50e-02	6.94e-18	2.23e-03	4.96e-02	×	7.29e+01
Esq4. CP3	5.03e-02	2.50e-02	2.34e-02	6.14e-02	✓	1.77e+03	**1.78e-04**	3.92e-04	**3.64e-06**	1.51e-03	✓	4.90e-01	5.29e-02	5.03e-03	3.04e-02	5.72e-02	✓	6.20e+01
Esq4. CP4	2.23e-02	1.01e-02	1.74e-02	8.12e-02	×	1.86e+03	**1.16e-04**	5.04e-04	**3.30e-06**	1.60e-03	✓	4.45e-01	4.29e-03	5.67e-03	4.06e-03	1.74e-02	×	9.18e+01

Tabla A.3: Mediana (Med), *Interquartile Range* (IQR), Mínimo (Min) y Máximo (Max) de la distribución de RMSE para el conjunto de datos de validación, junto a la indicación de éxito en la tarea de descubrir la ecuación original, y tiempo promedio en segundos (t_{avg}) para cada ecuación y algoritmo.

Función	Friedman Test Statistic	P-Valor
Vladislavleva		
1	×	3.50e-01
2	✓	2.94e-05
3	✓	1.33e-07
4	✓	4.89e-05
5	✓	3.81e-07
6	✓	1.45e-02
7	✓	5.10e-11
8	✓	3.81e-07
Feynman		
I.6.2a	✓	6.00e-10
I.6.2	✓	3.38e-11
I.6.2b	✓	5.10e-11
I.8.14	✓	3.42e-10
I.9.18	✓	2.67e-10
I.10.7	✓	2.42e-07
I.11.19	✓	1.64e-10
I.12.1	✓	1.04e-12
I.12.2	✓	3.27e-07
I.12.4	✓	1.87e-11
I.12.5	✓	1.42e-12
I.12.11	✓	1.62e-07
I.13.4	✓	4.58e-08
I.13.12	✓	5.06e-04
I.14.3	✓	3.78e-11
I.14.4	✓	1.94e-06
I.15.3x	✓	2.65e-09
I.15.3t	✓	1.09e-07
I.15.1	✓	2.53e-08
I.16.6	✓	5.82e-05
I.18.4	✓	1.44e-11
I.18.12	✓	2.59e-06
I.18.14	✓	6.63e-05
I.24.6	✓	4.92e-10
I.25.13	✓	3.76e-12
I.26.2	✓	5.07e-11
I.27.6	✓	1.41e-06
I.29.4	✓	2.25e-12
I.29.16	✓	1.26e-07
I.30.3	✓	2.11e-11
I.30.5	✓	2.15e-03
I.32.5	✓	2.24e-11
I.32.17	✓	3.27e-07
I.34.8	✓	4.14e-05

Función	**Friedman Test Statistic**	**P-Valor**
I.34.1	×	1.30e-01
I.34.14	✓	6.61e-10
I.34.27	✓	4.28e-05
I.37.4	✓	2.97e-09
I.38.12	✓	1.23e-11
I.39.1	✓	4.39e-05
I.39.11	✓	3.26e-06
I.39.22	✓	3.33e-05
I.40.1	✓	3.94e-08
I.41.16	✓	5.10e-11
I.43.16	✓	4.85e-06
I.43.31	✓	2.18e-11
I.43.43	✓	3.42e-10
I.44.4	×	1.90e-01
I.47.23	✓	3.68e-08
I.48.2	✓	9.36e-14
I.50.26	✓	3.50e-02
II.2.42	✓	6.58e-07
II.3.24	✓	5.42e-09
II.4.23	✓	1.99e-08
II.6.11	✓	1.28e-09
II.6.15a	✓	1.09e-07
II.6.15b	✓	2.70e-10
II.8.7	✓	9.28e-11
II.8.31	✓	1.91e-06
II.10.9	✓	2.32e-02
II.11.3	✓	1.84e-09
II.11.17	✓	1.30e-10
II.11.20	✓	1.64e-10
II.11.27	✓	1.48e-10
II.11.28	✓	1.58e-11
II.13.17	✓	1.10e-11
II.13.23	✓	3.85e-04
II.13.34	✓	4.76e-11
II.15.4	✓	2.56e-04
II.15.5	✓	4.50e-02
II.21.32	✓	2.50e-06
II.24.17	✓	3.42e-10
II.27.16	✓	4.62e-09
II.27.18	✓	3.14e-12
II.34.2a	✓	3.57e-07
II.34.2	✓	2.24e-03
II.34.11	✓	2.70e-10
II.34.29a	✓	1.69e-10
II.34.29b	✓	3.00e-12

Función	**Friedman Test Statistic**	**P-Valor**
II.35.18	✓	1.01e-10
II.35.21	✓	9.36e-14
II.36.38	✓	3.64e-10
II.37.1	✓	1.88e-08
II.38.3	✓	2.31e-04
II.38.14	✓	1.64e-10
III.4.32	✓	4.00e-08
III.4.33	✓	5.10e-11
III.7.38	✓	1.06e-07
III.8.54	×	1.31e-01
III.9.52	✓	2.70e-10
III.10.19	✓	1.14e-10
III.12.43	✓	1.54e-04
III.13.18	✓	3.00e-12
III.14.14	✓	1.21e-08
III.15.12	✓	5.00e-04
III.15.14	✓	3.30e-11
III.15.27	✓	5.39e-07
III.17.37	✓	2.65e-03
III.19.51	✓	2.59e-09
III.21.20	×	6.19e-01
Funciones IPQ		
Esq. 1 CP1	✓	3.44e-08
Esq. 1 CP2	×	8.45e-01
Esq. 1 CP3	✓	6.31e-11
Esq. 1 CP4	✓	1.14e-11
Esq. 2 CP1	✓	8.52e-11
Esq. 2 CP2	✓	7.67e-11
Esq. 2 CP3	✓	6.72e-11
Esq. 2 CP4	✓	2.00e-11
Esq. 3 CP1	✓	4.95e-11
Esq. 3 CP2	✓	6.31e-11
Esq. 3 CP3	✓	1.01e-10
Esq. 3 CP4	✓	1.44e-11
Esq. 4 CP1	✓	7.33e-10
Esq. 4 CP2	✓	6.25e-11
Esq. 4 CP3	✓	1.01e-10
Esq. 4 CP4	✓	9.36e-14

Tabla A.4: Resultados del Test de Friedman para las diferentes funciones y algoritmos

A.6. Boxplots correspondientes al *benchmark* de Vladislavleva

Diagramas de cajas para la distribución del RMSE en los puntos de validación para las ecuaciones del *benchmark* Vladislavleva. El símbolo ∘ representa la media. La caja muestra el rango intercuartílico (IQR), donde:

- ∘ : Media
- Línea intermedia : Mediana.
- Línea inferior : Primer cuartil (Q1),
- Línea superior : Tercer cuartil (Q3),
- ♢ : Outliers

Las barras (bigotes) representan los valores mínimo y máximo dentro del rango intercuartílico, excluyendo los valores atípicos. Los rombos grises indican los valores atípicos, que se encuentran fuera de los límites de $Q1 - 1{,}5 \times \text{IQR}$ y $Q3 + 1{,}5 \times \text{IQR}$.

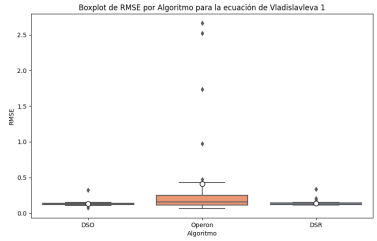

Figura A.1: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 1

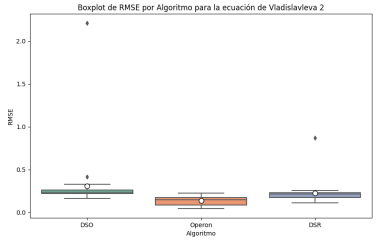

Figura A.2: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 2

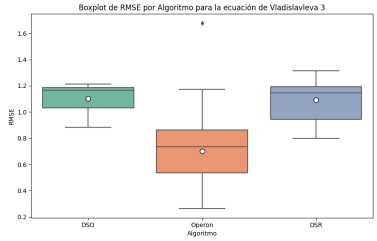

Figura A.3: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 3

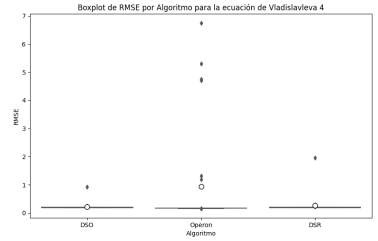

Figura A.4: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 4

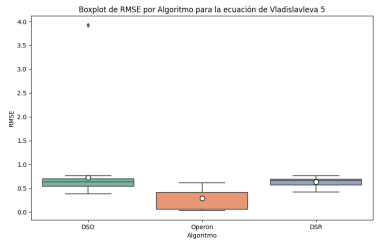

Figura A.5: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 5

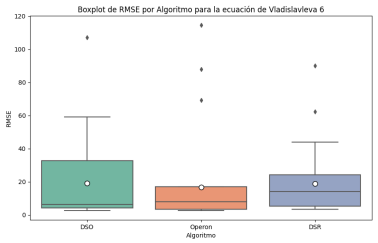

Figura A.6: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 6

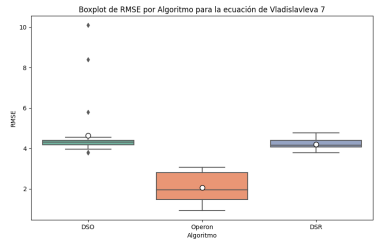

Figura A.7: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 7

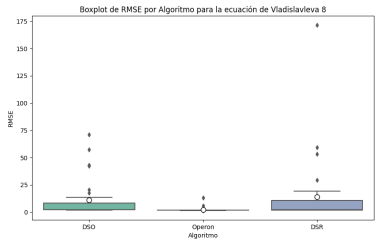

Figura A.8: Boxplots de RMSE por algoritmo para la ecuación de Vladislavleva 8

A.7. Boxplots correspondientes al *benchmark* de Feynman

Diagramas de cajas para la distribución del RMSE en los puntos de validación para las ecuaciones del *benchmark* Feynman. El símbolo ∘ representa la media. La caja muestra el rango intercuartílico (IQR), donde:

- ∘ : Media
- Línea intermedia : Mediana.
- Línea inferior : Primer cuartil (Q1),
- Línea superior : Tercer cuartil (Q3),
- ♢ : Outliers

Las barras (bigotes) representan los valores mínimo y máximo dentro del rango intercuartílico, excluyendo los valores atípicos. Los rombos grises indican los valores atípicos, que se encuentran fuera de los límites de $Q1 - 1{,}5 \times \text{IQR}$ y $Q3 + 1{,}5 \times \text{IQR}$.

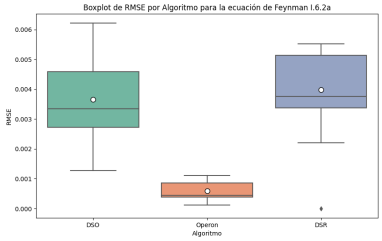

Figura A.9: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.6.2a

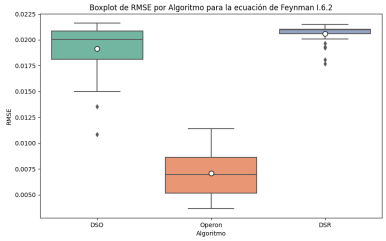

Figura A.10: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.6.2

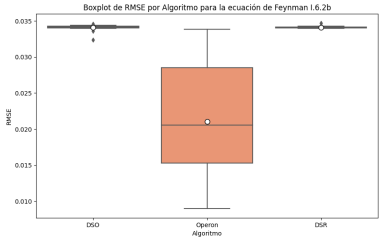

Figura A.11: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.6.2b

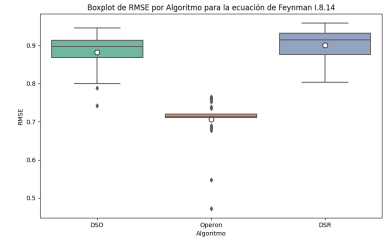

Figura A.12: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.8.14

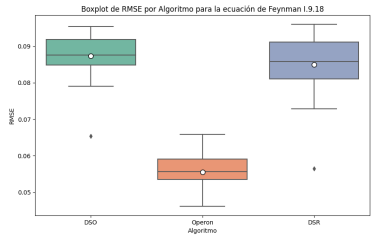

Figura A.13: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.9.18

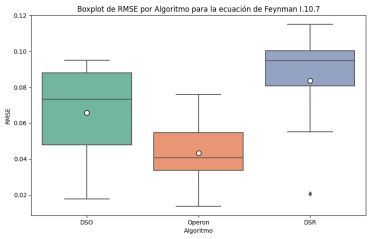

Figura A.14: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.10.7

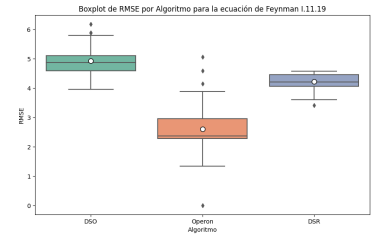

Figura A.15: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.11.19

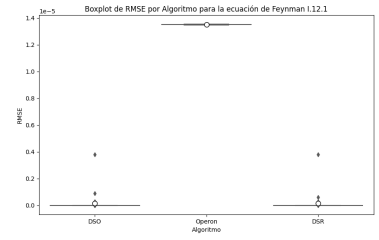

Figura A.16: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.12.1

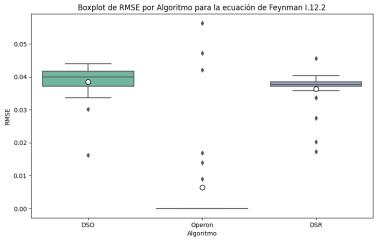

Figura A.17: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.12.2

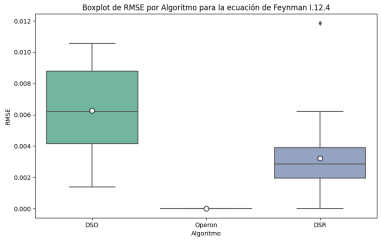

Figura A.18: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.12.4

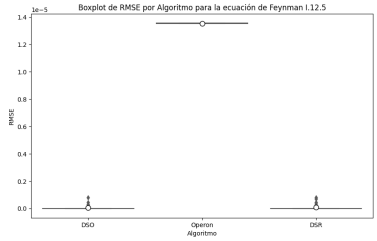

Figura A.19: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.12.5

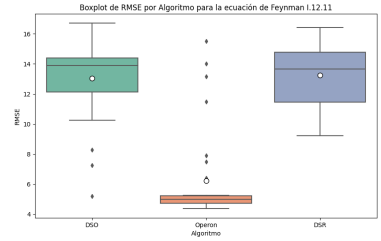

Figura A.20: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.12.11

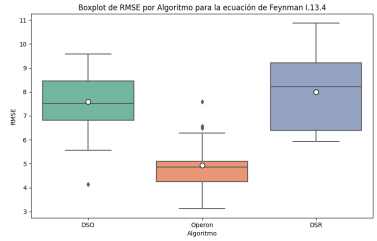

Figura A.21: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.13.4

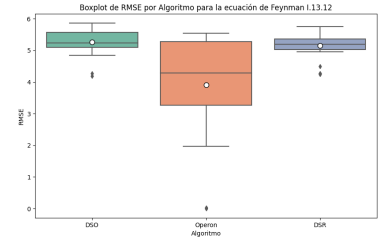

Figura A.22: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.13.12

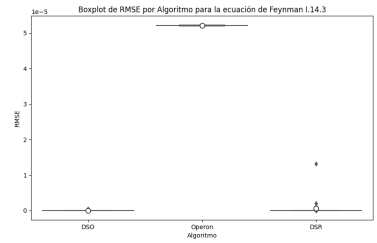

Figura A.23: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.14.3

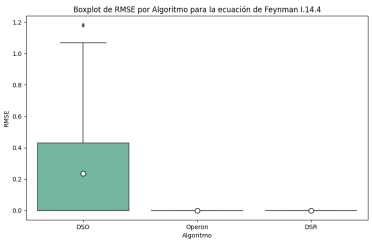

Figura A.24: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.14.4

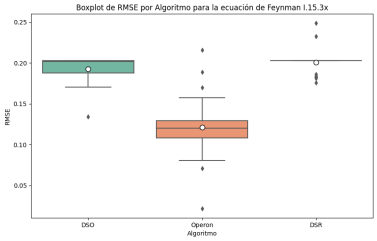

Figura A.25: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.15.3x

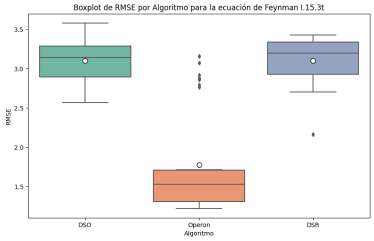

Figura A.26: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.15.3t

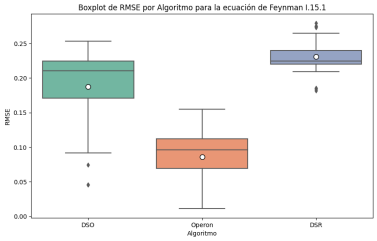

Figura A.27: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.15.1

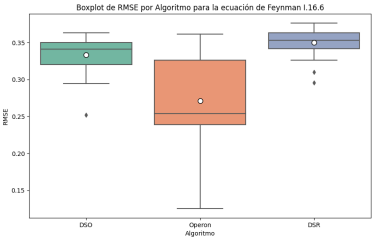

Figura A.28: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.16.6

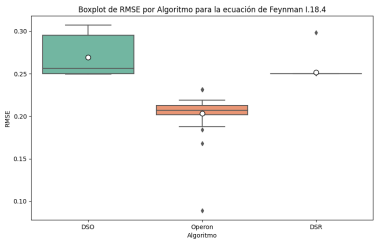

Figura A.29: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.18.4

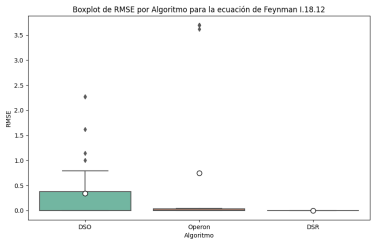

Figura A.30: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.18.12

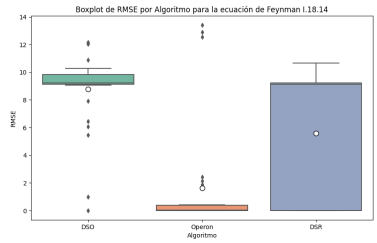

Figura A.31: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.18.14

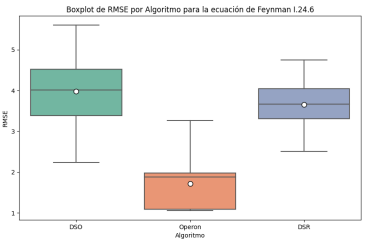

Figura A.32: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.24.6

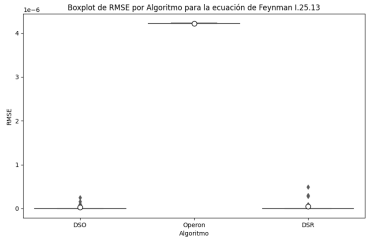

Figura A.33: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.25.13

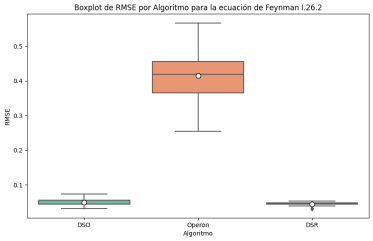

Figura A.34: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.26.2

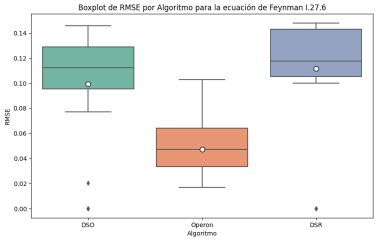

Figura A.35: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.27.6

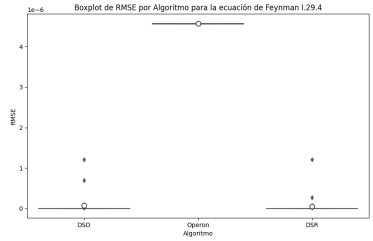

Figura A.36: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.29.4

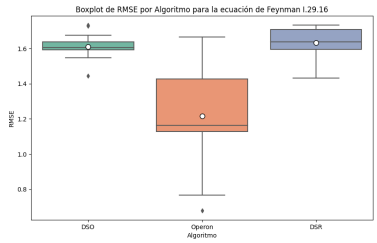

Figura A.37: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.29.16

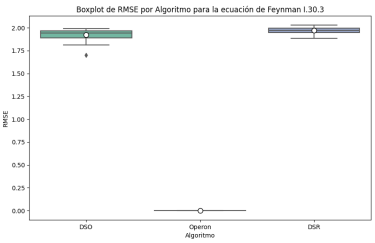

Figura A.38: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.30.3

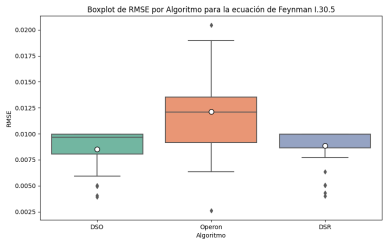

Figura A.39: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.30.5

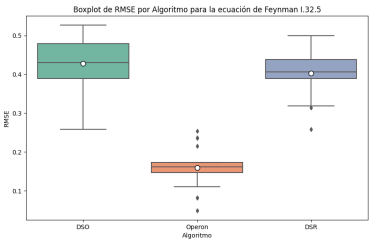

Figura A.40: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.32.5

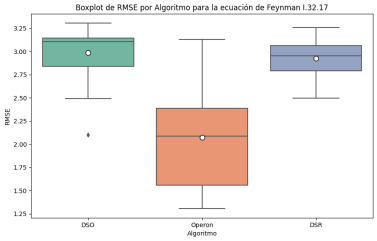

Figura A.41: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.32.17

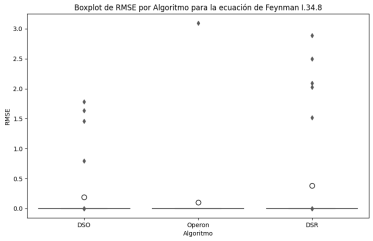

Figura A.42: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.34.8

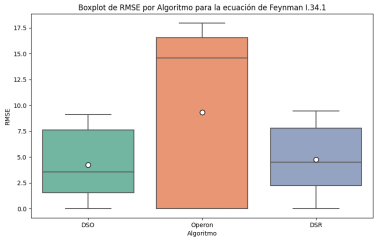

Figura A.43: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.34.1

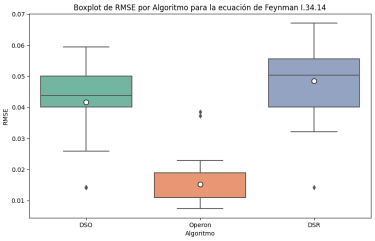

Figura A.44: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.34.14

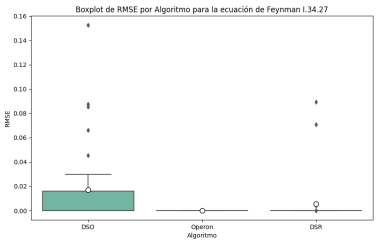

Figura A.45: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.34.27

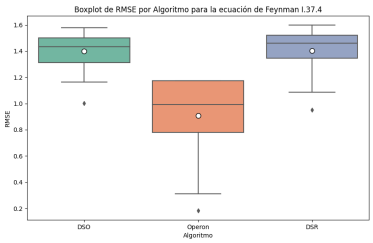

Figura A.46: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.37.4

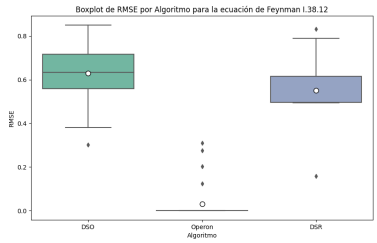

Figura A.47: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.38.12

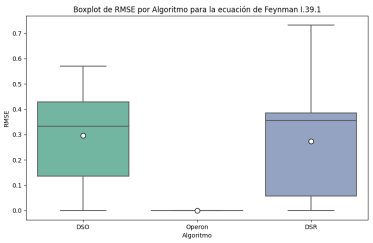

Figura A.48: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.39.1

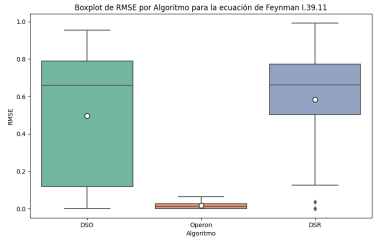

Figura A.49: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.39.11

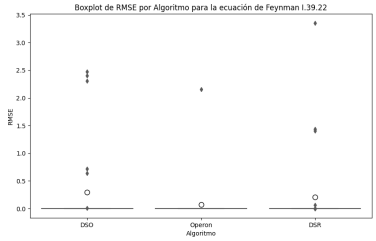

Figura A.50: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.39.22

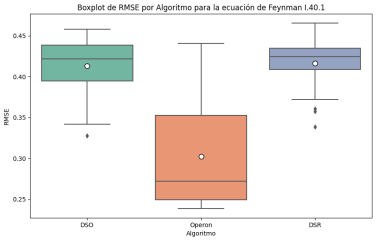

Figura A.51: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.40.1

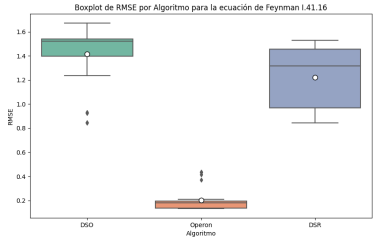

Figura A.52: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.41.16

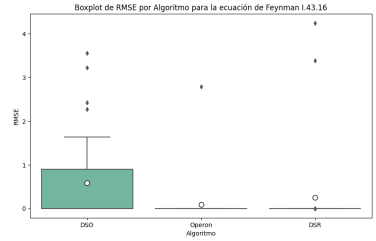

Figura A.53: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.43.16

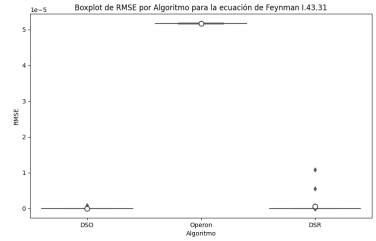

Figura A.54: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.43.31

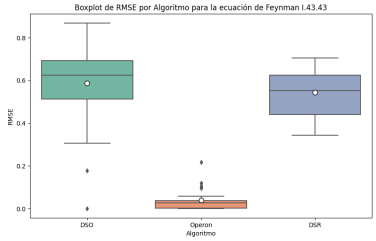

Figura A.55: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.43.43

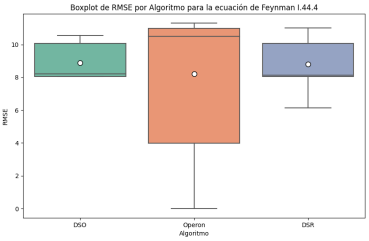

Figura A.56: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.44.4

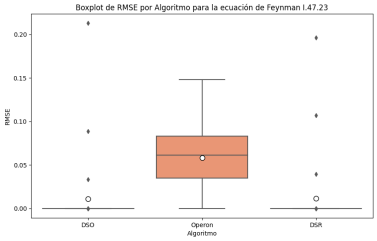

Figura A.57: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.47.23

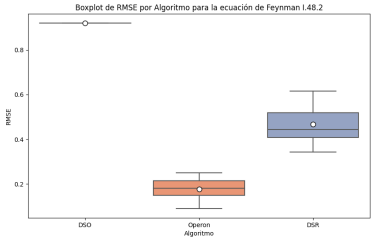

Figura A.58: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.48.2

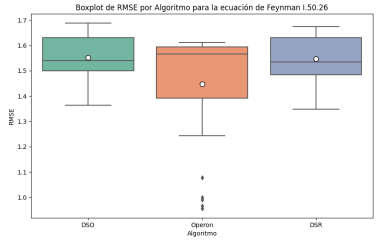

Figura A.59: Boxplots de RMSE por algoritmo para la ecuación de Feynman I.50.26

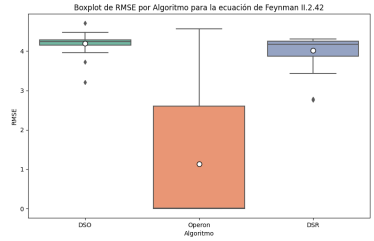

Figura A.60: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.2.42

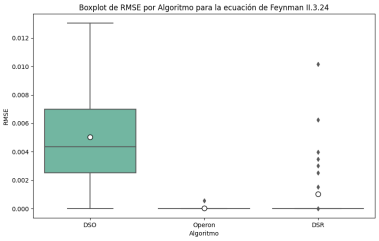

Figura A.61: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.3.24

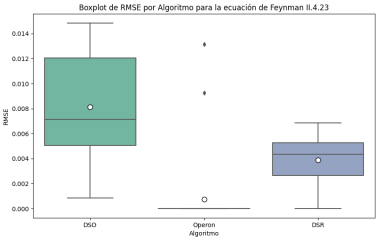

Figura A.62: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.4.23

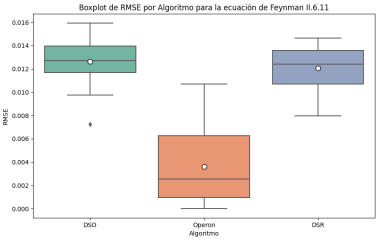

Figura A.63: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.6.11

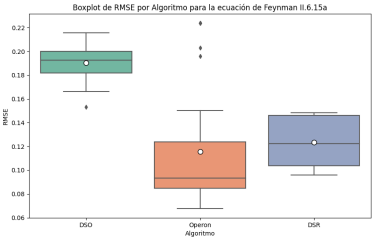

Figura A.64: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.6.15a

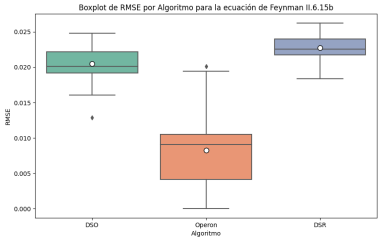

Figura A.65: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.6.15b

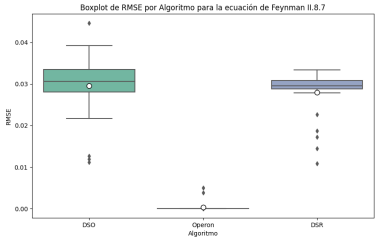

Figura A.66: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.8.7

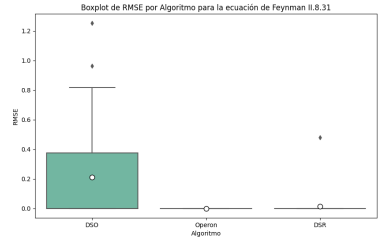

Figura A.67: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.8.31

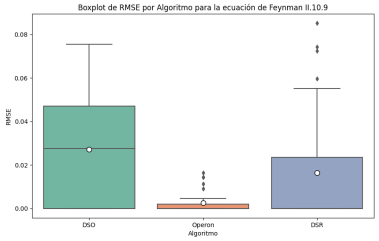

Figura A.68: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.10.9

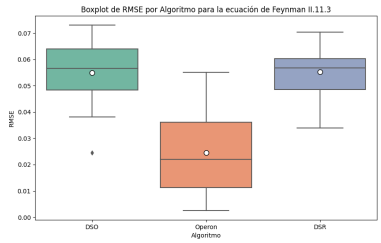

Figura A.69: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.11.3

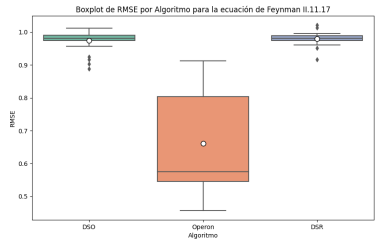

Figura A.70: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.11.17

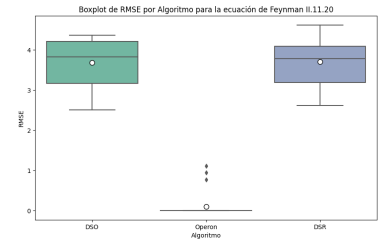

Figura A.71: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.11.20

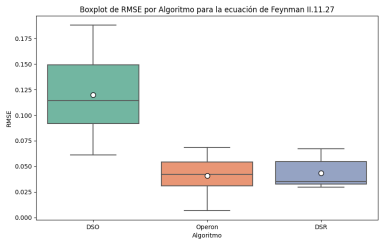

Figura A.72: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.11.27

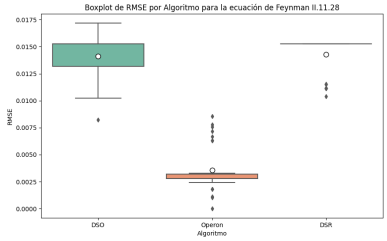

Figura A.73: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.11.28

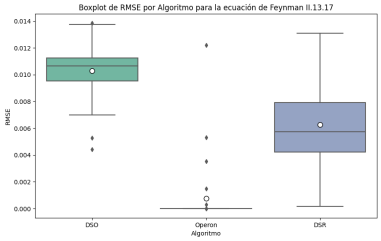

Figura A.74: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.13.17

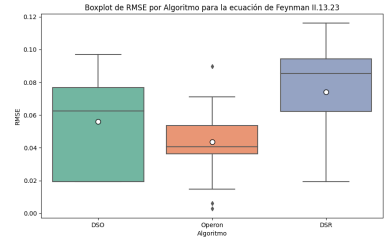

Figura A.75: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.13.23

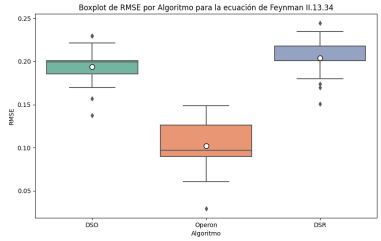

Figura A.76: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.13.34

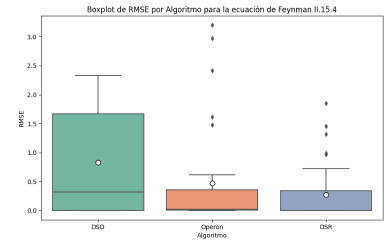

Figura A.77: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.15.4

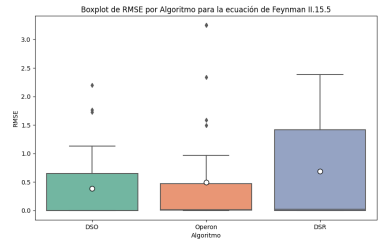

Figura A.78: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.15.5

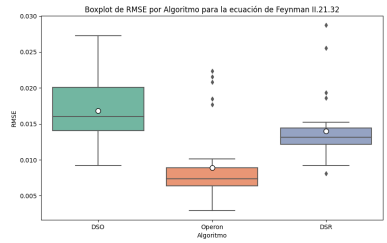

Figura A.79: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.21.32

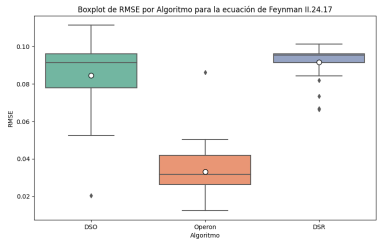

Figura A.80: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.24.17

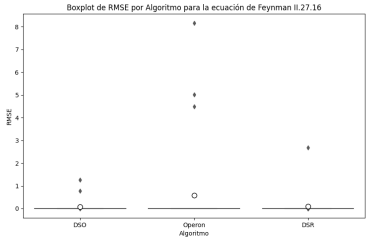

Figura A.81: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.27.16

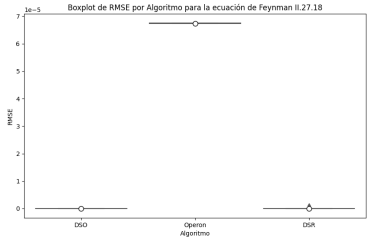

Figura A.82: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.27.18

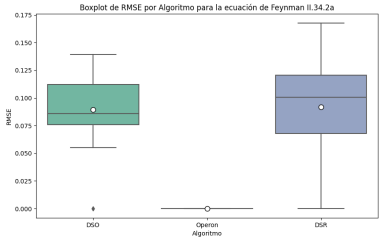

Figura A.83: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.34.2a

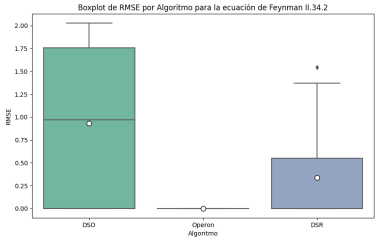

Figura A.84: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.34.2

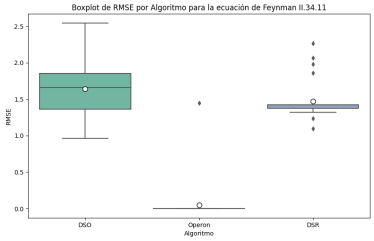

Figura A.85: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.34.11

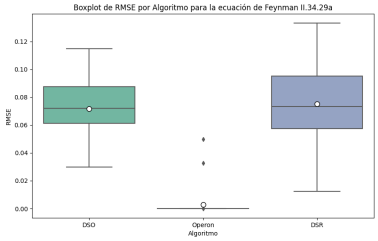

Figura A.86: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.34.29a

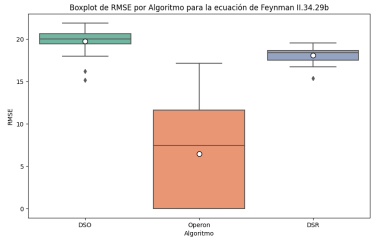

Figura A.87: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.34.29b

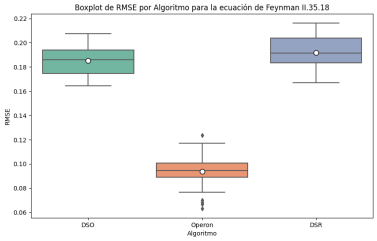

Figura A.88: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.35.18

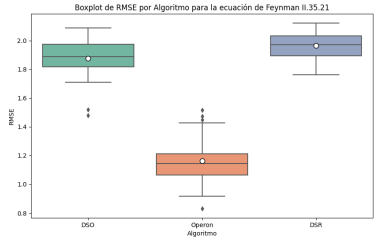

Figura A.89: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.35.21

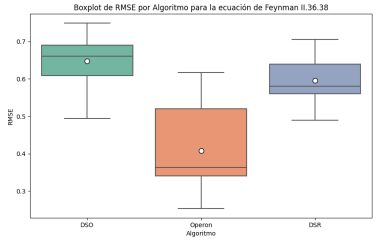

Figura A.90: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.36.38

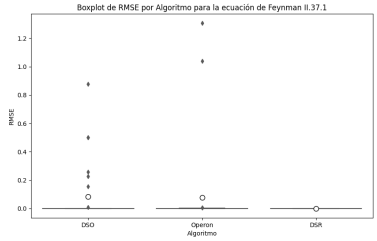

Figura A.91: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.37.1

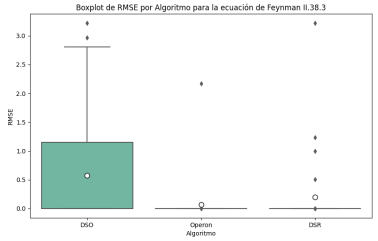

Figura A.92: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.38.3

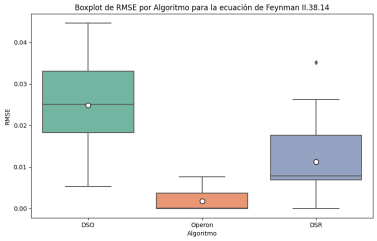

Figura A.93: Boxplots de RMSE por algoritmo para la ecuación de Feynman II.38.14

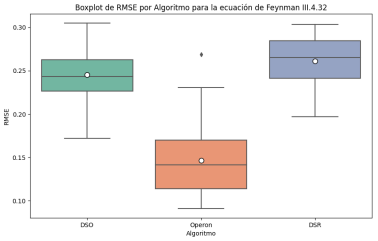

Figura A.94: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.4.32

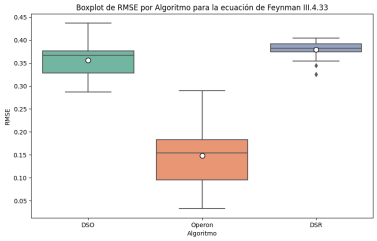

Figura A.95: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.4.33

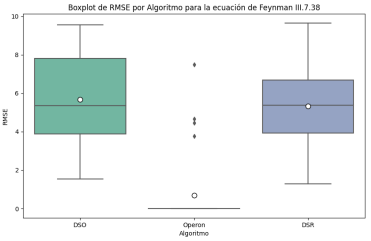

Figura A.96: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.7.38

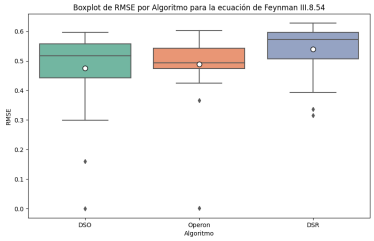

Figura A.97: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.8.54

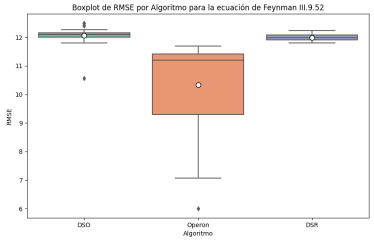

Figura A.98: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.9.52

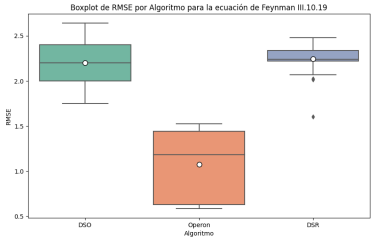

Figura A.99: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.10.19

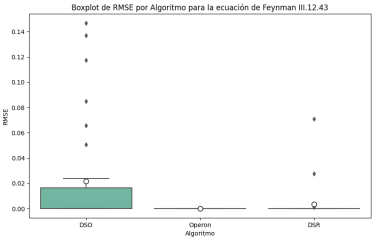

Figura A.100: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.12.43

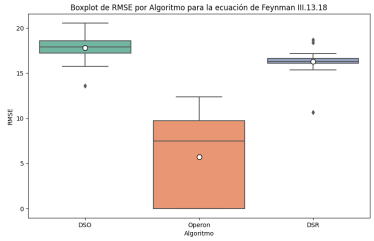

Figura A.101: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.13.18

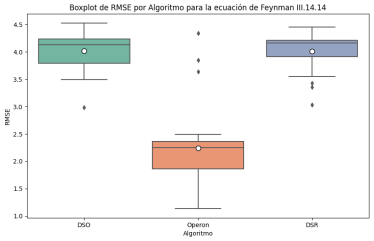

Figura A.102: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.14.14

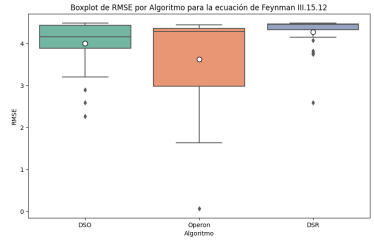

Figura A.103: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.15.12

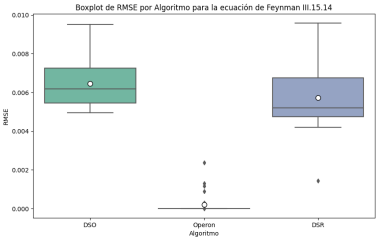

Figura A.104: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.15.14

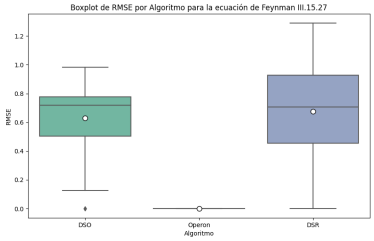

Figura A.105: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.15.27

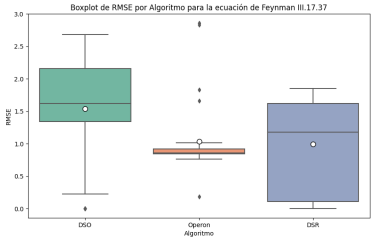

Figura A.106: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.17.37

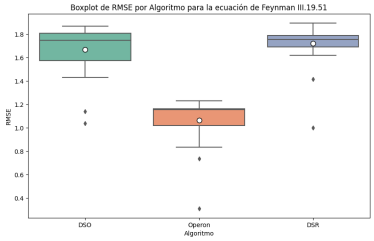

Figura A.107: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.19.51

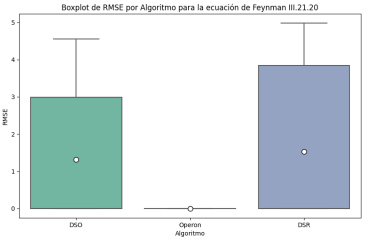

Figura A.108: Boxplots de RMSE por algoritmo para la ecuación de Feynman III.21.20

A.8. Boxplots correspondientes al *benchmark* de IPQ

Diagramas de cajas para la distribución del RMSE en los puntos de validación para las ecuaciones del *benchmark* de problemas de IPQ. El símbolo $\circ$ representa la media. La caja muestra el rango intercuartílico (IQR), donde:

- $\circ$: Media
- Línea intermedia : Mediana.
- Línea inferior : Primer cuartil (Q1),
- Línea superior : Tercer cuartil (Q3),
- $\diamondsuit$: Outliers

Las barras (bigotes) representan los valores mínimo y máximo dentro del rango intercuartílico, excluyendo los valores atípicos. Los rombos grises indican los valores atípicos, que se encuentran fuera de los límites de $Q1 - 1{,}5 \times \text{IQR}$ y $Q3 + 1{,}5 \times \text{IQR}$.

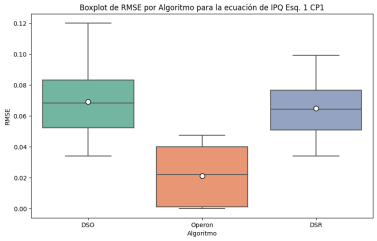

Figura A.109: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 1 CP1

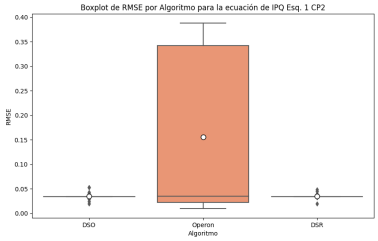

Figura A.110: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 1 CP2

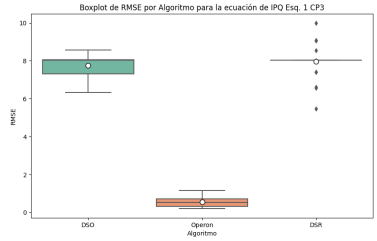

Figura A.111: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 1 CP3

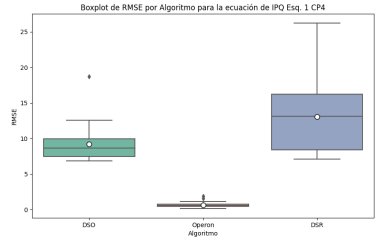

Figura A.112: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 1 CP4

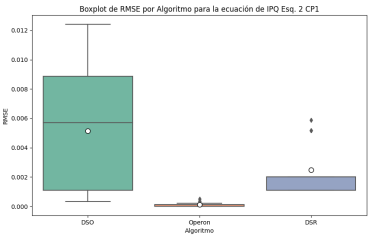

Figura A.113: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 2 CP1

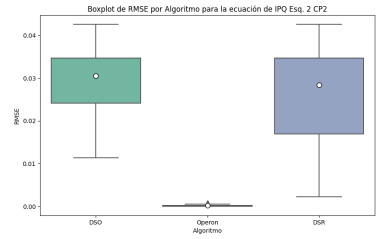

Figura A.114: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 2 CP2

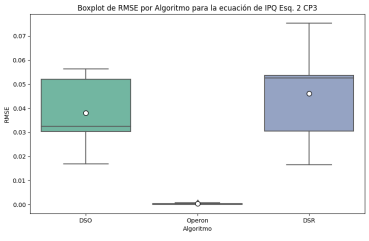

Figura A.115: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 2 CP3

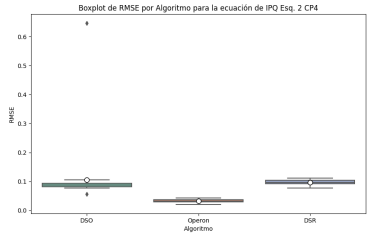

Figura A.116: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 2 CP4

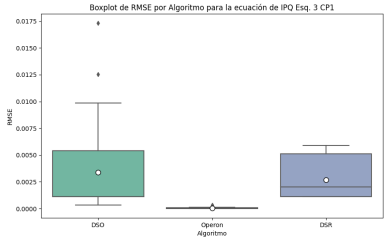

Figura A.117: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 3 CP1

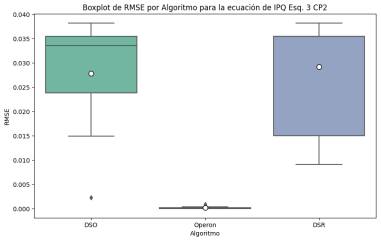

Figura A.118: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 3 CP2

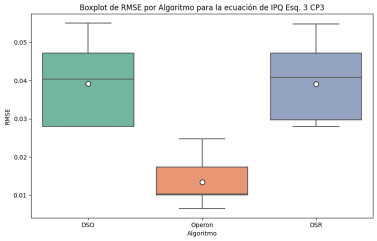

Figura A.119: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 3 CP3

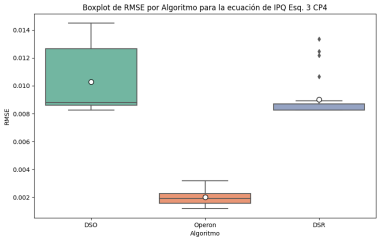

Figura A.120: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 3 CP4

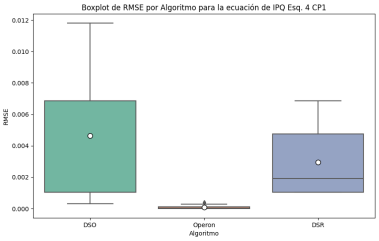

Figura A.121: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 4 CP1

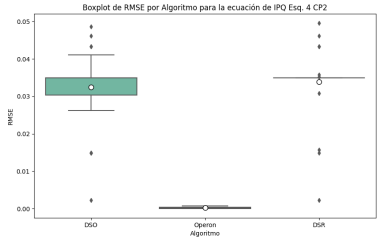

Figura A.122: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 4 CP2

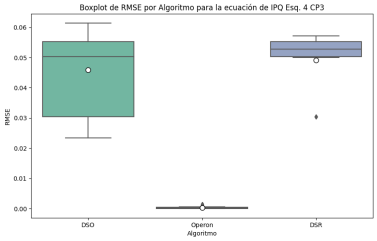

Figura A.123: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 4 CP3

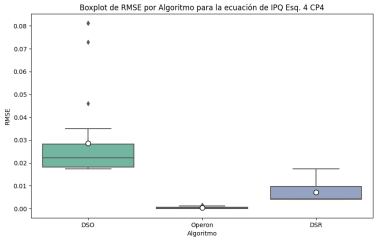

Figura A.124: Boxplots de RMSE por algoritmo para la ecuación de IPQ Esq. 4 CP4

A.9. Evaluación estadística de a pares por procedimiento *post-hoc* de Holm para el *benchmark* Vladislavleva

Resultados del procedimiento *post-hoc* de Holm para las configuraciones en las que la prueba de Friedman indicó que las distribuciones de valores de RMSE, al ejecutar los algoritmos DSO, Operon y DSR para las ecuaciones del *benchmark* Vladislavleva, son estadísticamente diferentes.

El procedimiento de Holm se utilizó para ajustar los valores p en las comparaciones múltiples entre los algoritmos, controlando la tasa de error de tipo I. En los casos donde se detectaron diferencias significativas, se consideró como el mejor algoritmo aquel con la menor mediana de RMSE.

Para interpretar las flechas en la tabla:

- El símbolo $\triangle$ indica que el algoritmo en la columna es mejor que el de la fila, es decir, que su mediana de RMSE es menor y que esta diferencia es estadísticamente significativa.
- El símbolo $\triangleleft$ indica que el algoritmo en la fila es mejor que el de la columna bajo los mismos criterios.

	Operon	DSR
DSO	$\triangle$ $1{,}87e-05$	– $9{,}33e-02$
Operon		$\triangleleft$ $9{,}02e-03$

Tabla A.5: Ecuación Vladislavleva 2.

	Operon	DSR
DSO	$\triangle$ $7{,}25e-07$	– $5{,}19e-01$
Operon		$\triangleleft$ $1{,}25e-05$

Tabla A.6: Ecuación Vladislavleva 3.

	Operon	DSR
DSO	$\triangle$ $9{,}02e-03$	– $1{,}97e-01$
Operon		$\triangleleft$ $1{,}08e-04$

Tabla A.7: Ecuación Vladislavleva 4.

	Operon	DSR
DSO	$\triangle$ $4{,}84e-07$	– $8{,}97e-01$
Operon		$\triangleleft$ $3{,}61e-07$

Tabla A.8: Ecuación Vladislavleva 5.

	Operon	DSR
DSO	– $7{,}32e-01$	– $7{,}32e-01$
Operon		– $2{,}12e-01$

Tabla A.9: Ecuación Vladislavleva 6.

	Operon	DSR
DSO	$\triangle$ $1{,}37e-10$	– $1{,}21e-01$
Operon		$\triangleleft$ $9{,}56e-07$

Tabla A.10: Ecuación Vladislavleva 7.

	Operon	DSR
DSO	$\triangle$ $2{,}79e-06$	– $6{,}06e-01$
Operon		$\triangleleft$ $2{,}27e-05$

Tabla A.11: Ecuación Vladislavleva 8.

A.10. Evaluación estadística de a pares por procedimiento *post-hoc* de Holm para el *benchmark* Feynman

Resultados del procedimiento *post-hoc* de Holm para las configuraciones en las que la prueba de Friedman indicó que las distribuciones de valores de RMSE, al ejecutar los algoritmos DSO, Operon y DSR para las ecuaciones del *benchmark* Feynman, son estadísticamente diferentes.

El procedimiento de Holm se utilizó para ajustar los valores p en las comparaciones múltiples entre los algoritmos, controlando la tasa de error de tipo I. En los casos donde se detectaron diferencias significativas, se consideró como el mejor algoritmo aquel con la menor mediana de RMSE.

Para interpretar las flechas en la tabla:

- El símbolo $\triangle$ indica que el algoritmo en la columna es mejor que el de la fila, es decir, que su mediana de RMSE es menor y que esta diferencia es estadísticamente significativa.
- El símbolo $\triangleleft$ indica que el algoritmo en la fila es mejor que el de la columna bajo los mismos criterios.

	Operon	DSR
DSO	$\triangle$ $2{,}41e-07$	– $5{,}19e-01$
Operon		$\triangleleft$ $8{,}63e-09$

Tabla A.12: Ecuación Feynman I.6.2a.

	Operon	DSR
DSO	$\triangle$ $1{,}34e-06$	– $9{,}33e-02$
Operon		$\triangleleft$ $8{,}88e-11$

Tabla A.13: Ecuación Feynman I.6.2.

	Operon	DSR
DSO	$\triangle$ $1{,}37e-10$	– $1{,}21e-01$
Operon		$\triangleleft$ $9{,}56e-07$

Tabla A.14: Ecuación Feynman I.6.2b.

	Operon	DSR
DSO	$\triangle$ $4{,}84e-07$	– $3{,}66e-01$
Operon		$\triangleleft$ $3{,}89e-09$

Tabla A.15: Ecuación Feynman I.8.14.

	Operon	DSR
DSO	$\triangle$ $1{,}15e-09$	– $1{,}97e-01$
Operon		$\triangleleft$ $1{,}34e-06$

Tabla A.16: Ecuación Feynman I.9.18.

	Operon	DSR
DSO	$\triangle$ $4{,}83e-03$	$\triangleleft$ $1{,}42e-02$
Operon		$\triangleleft$ $1{,}23e-07$

Tabla A.17: Ecuación Feynman I.10.7.

	Operon	DSR
DSO	$\triangle$ $5{,}71e-11$	$\triangle$ $1{,}58e-03$
Operon		$\triangleleft$ $1{,}58e-03$

Tabla A.18: Ecuación Feynman I.11.19.

	Operon	DSR
DSO	$\triangleleft$ $8{,}63e-09$	– $7{,}96e-01$
Operon		$\triangle$ $2{,}69e-08$

Tabla A.19: Ecuación Feynman I.12.1.

	Operon	DSR
DSO	$\triangle$ $7{,}25e-07$	– $3{,}02e-01$
Operon		$\triangleleft$ $7{,}22e-05$

Tabla A.20: Ecuación Feynman I.12.2.

	Operon	DSR
DSO	$\triangle$ $9{,}42e-12$	$\triangle$ $6{,}71e-03$
Operon		$\triangleleft$ $4{,}08e-05$

Tabla A.21: Ecuación Feynman I.12.4.

	Operon	DSR
DSO	$\triangleleft$ $1{,}28e-08$	– $8{,}97e-01$
Operon		$\triangle$ $1{,}84e-08$

Tabla A.22: Ecuación Feynman I.12.5.

	Operon	DSR
DSO	$\triangle$ $3{,}56e-06$	– $8{,}97e-01$
Operon		$\triangleleft$ $2{,}79e-06$

Tabla A.23: Ecuación Feynman I.12.11.

	Operon	DSR
DSO	△ $1{,}43e-06$	– $1{,}00e+00$
Operon		◁ $1{,}43e-06$

Tabla A.24: Ecuación Feynman I.13.4.

	Operon	DSR
DSO	△ $1{,}43e-04$	– $4{,}39e-01$
Operon		◁ $1{,}99e-03$

Tabla A.25: Ecuación Feynman I.13.12.

	Operon	DSR
DSO	◁ $1{,}73e-09$	– $4{,}39e-01$
Operon		△ $1{,}18e-07$

Tabla A.26: Ecuación Feynman I.14.3.

	Operon	DSR
DSO	– $1{,}97e-01$	△ $6{,}01e-04$
Operon		△ $2{,}79e-06$

Tabla A.27: Ecuación Feynman I.14.4.

	Operon	DSR
DSO	△ $1{,}86e-06$	– $5{,}19e-01$
Operon		◁ $8{,}51e-08$

Tabla A.28: Ecuación Feynman I.15.3x.

	Operon	DSR
DSO	△ $2{,}27e-05$	– $3{,}66e-01$
Operon		◁ $3{,}61e-07$

Tabla A.29: Ecuación Feynman I.15.3t.

	Operon	DSR
DSO	△ $6{,}72e-06$	– $4{,}39e-01$
Operon		◁ $1{,}77e-07$

Tabla A.30: Ecuación Feynman I.15.1.

	Operon	DSR
DSO	△ $1{,}63e-02$	– $9{,}33e-02$
Operon		◁ $4{,}58e-05$

Tabla A.31: Ecuación Feynman I.16.6.

	Operon	DSR
DSO	△ $2{,}12e-10$	– $1{,}56e-01$
Operon		◁ $6{,}81e-07$

Tabla A.32: Ecuación Feynman I.18.4.

	Operon	DSR
DSO	◁ $2{,}50e-03$	– $9{,}33e-02$
Operon		△ $2{,}79e-06$

Tabla A.33: Ecuación Feynman I.18.12.

	Operon	DSR
DSO	△ $6{,}13e-05$	△ $1{,}34e-02$
Operon		– $1{,}21e-01$

Tabla A.34: Ecuación Feynman I.18.14.

	Operon	DSR
DSO	△ $3{,}89e-09$	– $3{,}66e-01$
Operon		◁ $4{,}84e-07$

Tabla A.35: Ecuación Feynman I.24.6.

	Operon	DSR
DSO	$\triangleleft$ $1{,}73e-09$	– $4{,}39e-01$
Operon		$\triangle$ $1{,}18e-07$

Tabla A.36: Ecuación Feynman I.25.13.

	Operon	DSR
DSO	$\triangleleft$ $3{,}05e-05$	– $1{,}56e-01$
Operon		$\triangle$ $2{,}76e-08$

Tabla A.37: Ecuación Feynman I.26.2.

	Operon	DSR
DSO	$\triangle$ $5{,}44e-05$	– $6{,}06e-01$
Operon		$\triangleleft$ $7{,}35e-06$

Tabla A.38: Ecuación Feynman I.27.6.

	Operon	DSR
DSO	$\triangleleft$ $3{,}91e-08$	– $6{,}99e-01$
Operon		$\triangle$ $5{,}81e-09$

Tabla A.39: Ecuación Feynman I.29.4.

	Operon	DSR
DSO	$\triangle$ $9{,}17e-06$	– $6{,}06e-01$
Operon		$\triangleleft$ $1{,}02e-06$

Tabla A.40: Ecuación Feynman I.29.16.

	Operon	DSR
DSO	$\triangle$ $2{,}58e-06$	– $5{,}28e-02$
Operon		$\triangleleft$ $3{,}66e-11$

Tabla A.41: Ecuación Feynman I.30.3.

	Operon	DSR
DSO	$\triangleleft$ $3{,}75e-03$	– $5{,}19e-01$
Operon		$\triangle$ $1{,}96e-02$

Tabla A.42: Ecuación Feynman I.30.5.

	Operon	DSR
DSO	$\triangle$ $8{,}88e-11$	– $9{,}33e-02$
Operon		$\triangleleft$ $1{,}34e-06$

Tabla A.43: Ecuación Feynman I.32.5.

	Operon	DSR
DSO	$\triangle$ $7{,}25e-07$	– $3{,}02e-01$
Operon		$\triangleleft$ $7{,}22e-05$

Tabla A.44: Ecuación Feynman I.32.17.

	Operon	DSR
DSO	$\triangleleft$ $2{,}47e-04$	– $8{,}97e-01$
Operon		$\triangle$ $2{,}80e-04$

Tabla A.45: Ecuación Feynman I.34.8.

	Operon	DSR
DSO	$\triangle$ $3{,}56e-06$	– $1{,}97e-01$
Operon		$\triangleleft$ $3{,}89e-09$

Tabla A.46: Ecuación Feynman I.34.14.

	Operon	DSR
DSO	$\triangleleft$ $2{,}36e-02$	– $5{,}28e-02$
Operon		$\triangle$ $2{,}53e-05$

Tabla A.47: Ecuación Feynman I.34.27.

	Operon	DSR
DSO	△ $2{,}41e-07$	– $7{,}96e-01$
Operon		◁ $8{,}51e-08$

Tabla A.48: Ecuación Feynman I.37.4.

	Operon	DSR
DSO	△ $1{,}49e-11$	△ $2{,}82e-02$
Operon		◁ $4{,}90e-06$

Tabla A.49: Ecuación Feynman I.38.12.

	Operon	DSR
DSO	△ $3{,}41e-05$	– $1{,}56e-01$
Operon		◁ $5{,}97e-03$

Tabla A.50: Ecuación Feynman I.39.1.

	Operon	DSR
DSO	△ $3{,}62e-04$	– $3{,}02e-01$
Operon		◁ $5{,}35e-06$

Tabla A.51: Ecuación Feynman I.39.11.

	Operon	DSR
DSO	◁ $7{,}35e-03$	– $1{,}21e-01$
Operon		△ $2{,}53e-05$

Tabla A.52: Ecuación Feynman I.39.22.

	Operon	DSR
DSO	△ $1{,}34e-06$	– $8{,}97e-01$
Operon		◁ $1{,}02e-06$

Tabla A.53: Ecuación Feynman I.40.1.

	Operon	DSR
DSO	△ $1{,}37e-10$	– $1{,}21e-01$
Operon		◁ $9{,}56e-07$

Tabla A.54: Ecuación Feynman I.41.16.

	Operon	DSR
DSO	◁ $2{,}83e-02$	△ $2{,}83e-02$
Operon		△ $2{,}79e-06$

Tabla A.55: Ecuación Feynman I.43.16.

	Operon	DSR
DSO	◁ $2{,}60e-09$	– $5{,}19e-01$
Operon		△ $8{,}19e-08$

Tabla A.56: Ecuación Feynman I.43.31.

	Operon	DSR
DSO	△ $3{,}89e-09$	– $3{,}66e-01$
Operon		◁ $4{,}84e-07$

Tabla A.57: Ecuación Feynman I.43.43.

	Operon	DSR
DSO	◁ $5{,}13e-07$	– $6{,}99e-01$
Operon		△ $2{,}58e-06$

Tabla A.58: Ecuación Feynman I.47.23.

	Operon	DSR
DSO	△ $2{,}85e-14$	△ $2{,}15e-04$
Operon		◁ $2{,}15e-04$

Tabla A.59: Ecuación Feynman I.48.2.

	Operon	DSR
DSO	$\triangleleft$ $3{,}55e-02$	$-$ $4{,}39e-01$
Operon		$-$ $1{,}63e-01$

Tabla A.60: Ecuación Feynman I.50.26.

	Operon	DSR
DSO	$\triangle$ $7{,}25e-07$	$-$ $1{,}56e-01$
Operon		$\triangleleft$ $3{,}62e-04$

Tabla A.61: Ecuación Feynman II.2.42.

	Operon	DSR
DSO	$\triangle$ $1{,}26e-04$	$\triangle$ $3{,}89e-09$
Operon		$\triangle$ $3{,}89e-02$

Tabla A.62: Ecuación Feynman II.3.24.

	Operon	DSR
DSO	$\triangle$ $8{,}63e-09$	$\triangle$ $9{,}82e-03$
Operon		$\triangleleft$ $1{,}58e-03$

Tabla A.63: Ecuación Feynman II.4.23.

	Operon	DSR
DSO	$\triangle$ $8{,}63e-09$	$-$ $3{,}02e-01$
Operon		$\triangleleft$ $1{,}86e-06$

Tabla A.64: Ecuación Feynman II.6.11.

	Operon	DSR
DSO	$\triangle$ $3{,}61e-07$	$\triangle$ $2{,}27e-05$
Operon		$-$ $3{,}66e-01$

Tabla A.65: Ecuación Feynman II.6.15a.

	Operon	DSR
DSO	$\triangle$ $1{,}86e-06$	$-$ $1{,}56e-01$
Operon		$\triangleleft$ $7{,}55e-10$

Tabla A.66: Ecuación Feynman II.6.15b.

	Operon	DSR
DSO	$\triangle$ $1{,}15e-09$	$-$ $3{,}66e-01$
Operon		$\triangleleft$ $1{,}69e-07$

Tabla A.67: Ecuación Feynman II.8.7.

	Operon	DSR
DSO	$\triangleleft$ $2{,}39e-02$	$\triangle$ $9{,}02e-03$
Operon		$\triangle$ $1{,}02e-06$

Tabla A.68: Ecuación Feynman II.8.31.

	Operon	DSR
DSO	$-$ $2{,}45e-01$	$\triangle$ $2{,}01e-02$
Operon		$-$ $2{,}43e-01$

Tabla A.69: Ecuación Feynman II.10.9.

	Operon	DSR
DSO	$\triangle$ $1{,}23e-07$	$-$ $8{,}97e-01$
Operon		$\triangleleft$ $1{,}69e-07$

Tabla A.70: Ecuación Feynman II.11.3.

	Operon	DSR
DSO	$\triangle$ $3{,}91e-08$	$-$ $6{,}99e-01$
Operon		$\triangleleft$ $5{,}81e-09$

Tabla A.71: Ecuación Feynman II.11.17.

	Operon	DSR
DSO	△ $8{,}63e-09$	– $7{,}96e-01$
Operon		◁ $2{,}69e-08$

Tabla A.72: Ecuación Feynman II.11.20.

	Operon	DSR
DSO	△ $5{,}67e-08$	△ $3{,}89e-09$
Operon		– $6{,}06e-01$

Tabla A.73: Ecuación Feynman II.11.27.

	Operon	DSR
DSO	△ $1{,}18e-07$	– $4{,}39e-01$
Operon		◁ $1{,}73e-09$

Tabla A.74: Ecuación Feynman II.11.28.

	Operon	DSR
DSO	△ $3{,}73e-12$	△ $7{,}89e-04$
Operon		◁ $3{,}62e-04$

Tabla A.75: Ecuación Feynman II.13.17.

	Operon	DSR
DSO	– $1{,}21e-01$	◁ $4{,}03e-02$
Operon		◁ $3{,}23e-04$

Tabla A.76: Ecuación Feynman II.13.23.

	Operon	DSR
DSO	△ $3{,}42e-07$	– $2{,}45e-01$
Operon		◁ $4{,}96e-10$

Tabla A.77: Ecuación Feynman II.13.34.

	Operon	DSR
DSO	– $7{,}96e-01$	△ $1{,}58e-03$
Operon		△ $9{,}02e-04$

Tabla A.78: Ecuación Feynman II.15.4.

	Operon	DSR
DSO	– $6{,}04e-02$	– $1{,}06e-01$
Operon		– $6{,}99e-01$

Tabla A.79: Ecuación Feynman II.15.5.

	Operon	DSR
DSO	△ $1{,}43e-06$	– $5{,}28e-02$
Operon		◁ $3{,}89e-03$

Tabla A.80: Ecuación Feynman II.21.32.

	Operon	DSR
DSO	△ $4{,}84e-07$	– $3{,}66e-01$
Operon		◁ $3{,}89e-09$

Tabla A.81: Ecuación Feynman II.24.17.

	Operon	DSR
DSO	◁ $2{,}58e-06$	– $8{,}46e-01$
Operon		△ $1{,}43e-06$

Tabla A.82: Ecuación Feynman II.27.16.

	Operon	DSR
DSO	◁ $4{,}96e-10$	– $2{,}45e-01$
Operon		△ $3{,}42e-07$

Tabla A.83: Ecuación Feynman II.27.18.

	Operon	DSR
DSO	$\triangle$ $1{,}69e-05$	$-$ $6{,}99e-01$
Operon		$\triangleleft$ $3{,}87e-06$

Tabla A.84: Ecuación Feynman II.34.2a.

	Operon	DSR
DSO	$-$ $1{,}06e-01$	$\triangle$ $1{,}47e-03$
Operon		$-$ $1{,}21e-01$

Tabla A.85: Ecuación Feynman II.34.2.

	Operon	DSR
DSO	$\triangle$ $7{,}55e-10$	$-$ $1{,}56e-01$
Operon		$\triangleleft$ $1{,}86e-06$

Tabla A.86: Ecuación Feynman II.34.11.

	Operon	DSR
DSO	$\triangle$ $1{,}88e-08$	$-$ $1{,}00e+00$
Operon		$\triangleleft$ $1{,}88e-08$

Tabla A.87: Ecuación Feynman II.34.29a.

	Operon	DSR
DSO	$\triangle$ $1{,}45e-12$	$\triangle$ $4{,}51e-03$
Operon		$\triangleleft$ $2{,}27e-05$

Tabla A.88: Ecuación Feynman II.34.29b.

	Operon	DSR
DSO	$\triangle$ $5{,}67e-08$	$-$ $6{,}06e-01$
Operon		$\triangleleft$ $3{,}89e-09$

Tabla A.89: Ecuación Feynman II.35.18.

	Operon	DSR
DSO	$\triangle$ $6{,}72e-06$	$\triangleleft$ $2{,}01e-02$
Operon		$\triangleleft$ $9{,}42e-12$

Tabla A.90: Ecuación Feynman II.35.21.

	Operon	DSR
DSO	$\triangle$ $3{,}25e-10$	$\triangle$ $3{,}89e-02$
Operon		$\triangleleft$ $2{,}27e-05$

Tabla A.91: Ecuación Feynman II.36.38.

	Operon	DSR
DSO	$\triangleleft$ $3{,}89e-03$	$\triangle$ $6{,}71e-03$
Operon		$\triangle$ $1{,}88e-08$

Tabla A.92: Ecuación Feynman II.37.1.

	Operon	DSR
DSO	$\triangleleft$ $1{,}96e-02$	$-$ $1{,}56e-01$
Operon		$\triangle$ $1{,}88e-04$

Tabla A.93: Ecuación Feynman II.38.3.

	Operon	DSR
DSO	$\triangle$ $5{,}71e-11$	$\triangle$ $1{,}58e-03$
Operon		$\triangleleft$ $1{,}58e-03$

Tabla A.94: Ecuación Feynman II.38.14.

	Operon	DSR
DSO	$\triangle$ $3{,}56e-06$	$-$ $6{,}06e-01$
Operon		$\triangleleft$ $3{,}61e-07$

Tabla A.95: Ecuación Feynman III.4.32.

	Operon	DSR
DSO	△ $9{,}56e-07$	– $1{,}21e-01$
Operon		◁ $1{,}37e-10$

Tabla A.96: Ecuación Feynman III.4.33.

	Operon	DSR
DSO	△ $5{,}13e-07$	– $4{,}39e-01$
Operon		◁ $1{,}69e-05$

Tabla A.97: Ecuación Feynman III.7.38.

	Operon	DSR
DSO	△ $7{,}55e-10$	– $1{,}56e-01$
Operon		◁ $1{,}86e-06$

Tabla A.98: Ecuación Feynman III.9.52.

	Operon	DSR
DSO	△ $8{,}19e-08$	– $5{,}19e-01$
Operon		◁ $2{,}60e-09$

Tabla A.99: Ecuación Feynman III.10.19.

	Operon	DSR
DSO	◁ $2{,}83e-02$	– $9{,}33e-02$
Operon		△ $1{,}08e-04$

Tabla A.100: Ecuación Feynman III.12.43.

	Operon	DSR
DSO	△ $1{,}45e-12$	△ $4{,}51e-03$
Operon		◁ $2{,}27e-05$

Tabla A.101: Ecuación Feynman III.13.18.

	Operon	DSR
DSO	△ $3{,}61e-07$	– $8{,}97e-01$
Operon		◁ $4{,}84e-07$

Tabla A.102: Ecuación Feynman III.14.14.

	Operon	DSR
DSO	– $1{,}21e-01$	◁ $4{,}03e-02$
Operon		◁ $3{,}23e-04$

Tabla A.103: Ecuación Feynman III.15.12.

	Operon	DSR
DSO	△ $5{,}71e-11$	– $7{,}07e-02$
Operon		◁ $1{,}86e-06$

Tabla A.104: Ecuación Feynman III.15.14.

	Operon	DSR
DSO	△ $5{,}35e-06$	– $7{,}96e-01$
Operon		◁ $1{,}25e-05$

Tabla A.105: Ecuación Feynman III.15.27.

	Operon	DSR
DSO	△ $3{,}75e-03$	△ $1{,}96e-02$
Operon		– $5{,}19e-01$

Tabla A.106: Ecuación Feynman III.17.37.

	Operon	DSR
DSO	△ $1{,}69e-07$	– $8{,}97e-01$
Operon		◁ $1{,}23e-07$

Tabla A.107: Ecuación Feynman III.19.51.

A.11. Evaluación estadística de a pares por procedimiento *post-hoc* de Holm para el *benchmark* de problemas de IPQ

Resultados del procedimiento *post-hoc* de Holm para las configuraciones en las que la prueba de Friedman indicó que las distribuciones de valores de RMSE, al ejecutar los algoritmos DSO, Operon y DSR para las ecuaciones del *benchmark* de problemas de IPQ, son estadísticamente diferentes.

El procedimiento de Holm se utilizó para ajustar los valores p en las comparaciones múltiples entre los algoritmos, controlando la tasa de error de tipo I. En los casos donde se detectaron diferencias significativas, se consideró como el mejor algoritmo aquel con la menor mediana de RMSE.

Para interpretar las flechas en la tabla:

- El símbolo $\triangle$ indica que el algoritmo en la columna es mejor que el de la fila, es decir, que su mediana de RMSE es menor y que esta diferencia es estadísticamente significativa.
- El símbolo $\triangleleft$ indica que el algoritmo en la fila es mejor que el de la columna bajo los mismos criterios.

	Operon	DSR
DSO	△ $1{,}43e-06$	– $1{,}00e+00$
Operon		◁ $1{,}43e-06$

Tabla A.108: Ecuación IPQ Esq. 1 CP1.

	Operon	DSR
DSO	△ $1{,}84e-08$	– $8{,}97e-01$
Operon		◁ $1{,}28e-08$

Tabla A.109: Ecuación IPQ Esq. 1 CP3.

	Operon	DSR
DSO	△ $6{,}72e-06$	◁ $2{,}01e-02$
Operon		◁ $9{,}42e-12$

Tabla A.110: Ecuación IPQ Esq. 1 CP4.

	Operon	DSR
DSO	△ $1{,}73e-09$	– $4{,}39e-01$
Operon		◁ $1{,}18e-07$

Tabla A.111: Ecuación IPQ Esq. 2 CP1.

	Operon	DSR
DSO	△ $2{,}60e-09$	– $5{,}19e-01$
Operon		◁ $8{,}19e-08$

Tabla A.112: Ecuación IPQ Esq. 2 CP2.

	Operon	DSR
DSO	△ $2{,}41e-07$	– $3{,}02e-01$
Operon		◁ $7{,}55e-10$

Tabla A.113: Ecuación IPQ Esq. 2 CP3.

	Operon	DSR
DSO	△ $3{,}56e-06$	◁ $3{,}89e-02$
Operon		◁ $2{,}34e-11$

Tabla A.114: Ecuación IPQ Esq. 2 CP4.

	Operon	DSR
DSO	△ $4{,}84e-07$	– $1{,}97e-01$
Operon		◁ $3{,}25e-10$

Tabla A.115: Ecuación IPQ Esq. 3 CP1.

	Operon	DSR
DSO	△ $1{,}84e-08$	– $8{,}97e-01$
Operon		◁ $1{,}28e-08$

Tabla A.116: Ecuación IPQ Esq. 3 CP2.

	Operon	DSR
DSO	△ $5{,}67e-08$	– $6{,}06e-01$
Operon		◁ $3{,}89e-09$

Tabla A.117: Ecuación IPQ Esq. 3 CP3.

	Operon	DSR
DSO	△ $5{,}71e-11$	– $7{,}07e-02$
Operon		◁ $1{,}86e-06$

Tabla A.118: Ecuación IPQ Esq. 3 CP4.

	Operon	DSR
DSO	△ $5{,}67e-08$	– $8{,}97e-01$
Operon		◁ $4{,}03e-08$

Tabla A.119: Ecuación IPQ Esq. 4 CP1.

	Operon	DSR
DSO	$\triangle$ $1{,}69e-07$	– $3{,}66e-01$
Operon		$\triangleleft$ $1{,}15e-09$

Tabla A.120: Ecuación IPQ Esq. 4 CP2.

	Operon	DSR
DSO	$\triangle$ $5{,}67e-08$	– $6{,}06e-01$
Operon		$\triangleleft$ $3{,}89e-09$

Tabla A.121: Ecuación IPQ Esq. 4 CP3.

	Operon	DSR
DSO	$\triangle$ $2{,}85e-14$	$\triangle$ $2{,}15e-04$
Operon		$\triangleleft$ $2{,}15e-04$

Tabla A.122: Ecuación IPQ Esq. 4 CP4.