Práctica Electiva
Entrega 1

Profesores: Nicolás Hidalgo, Victor Reyes y Martín Gutiérrez
Estudiante: Joaquín Fernández
15 de Abril del 2023

Redes MLP y recurrentes para estimar el precondicionador Simplex en problemas de optimización continua

1. Resumen

El presente trabajo tiene como objetivo la revisión de investigaciones relacionadas con el uso de "Redes MLP y recurrentes para estimar el precondicionador Simplex en problemas de optimización continua". Se realizarán estudios para evaluar criterios, restricciones y conveniencias basados en soluciones binarias entregadas por un modelo simplex de carácter no lineal, que se encuentra sujeto a dominios continuos. Se abordarán "n" problemas que dependerán de diversos casos de conveniencia, como métricas de tiempo y consumo de recursos asociados con el hardware utilizado (tener criterios de parada), entre otros. El objetivo es aplicar un entrenamiento al modelo utilizando un agente de inteligencia artificial, en este caso, las redes MLP, que son técnicas de Deep Learning.

Es por ello que se tendrán investigaciones previas en documentos aledaños, como lo son anteproyectos, papers, tesis y artículos orientados al caso dentro de la comunidad científica.

2. Introducción

La optimización continua en matemáticas aplicadas es crucial para resolver problemas complejos en diversas áreas. Los problemas complejos no solo dependen del número de variables involucradas, sino también de los dominios de solución continuos, lo que puede generar soluciones inconsistentes. Para abordar este problema, se sugiere el uso de algoritmos resolutivos como el "Branch and Bound (B&B)" y el "Simplex". El B&B trabaja con intervalos de dominio y busca soluciones factibles, pero no es ágil en el tiempo. Mientras tanto, el Simplex es más rápido pero no es certero, por lo que se busca hacer una mezcla entre ambos algoritmos (pseudo-Simplex). El método Simplex es ampliamente utilizado para resolver problemas de optimización lineal y sus resultados pueden utilizarse para construir una matriz de precondicionadores para transformar el sistema original en uno optimizado. Además, se requiere el uso de una Multi Layer Perceptron (MLP) con datos de entrenamiento y variables de observación para validar los parámetros de entrada y estimar la solución óptima según los criterios involucrados.

Palabras claves

Dominios continuos, Método Simplex, MLP, modelos no lineales.

3. Estado del Arte

3.1. Ante proyecto Nicolás Calderón y Victor Reyes

El documento se enfoca en la predicción de precondicionadores pseudo-óptimos utilizando el método Simplex en problemas de optimización global. Estos problemas involucran la optimización de funciones no lineales con variables sujetas a dominios continuos acotados y restricciones no lineales.

¿Qué son los precondicionadores para este contexto?

En este contexto, los precondicionadores son los cambios realizados por el método Simplex en las cotas de las variables del problema. Estos cambios pueden combinarse linealmente para formar una matriz de precondicionadores. Dicha matriz resultante se utiliza para optimizar el sistema original del problema y reducir el tiempo de ejecución.

Enfoque en Simplex y Redes Neuronales MLP

El enfoque del documento se basa en investigaciones anteriores que han utilizado precondicionadores (criterios de evaluación) en otros métodos de optimización y redes neuronales para predecir el tiempo de ejecución del método Simplex. Las ANN son redes computacionales compuestas por "neuronas", éstas actúan como unidades de procesamiento y se definen mediante funciones matemáticas. Estas neuronas reciben una entrada, la procesan mediante una función de activación y generan una salida. Por lo tanto, las redes neuronales pueden proporcionar predicciones sobre la eficiencia y la aplicabilidad del método Simplex en sistemas lineales específicos.

En particular, se destaca la importancia de una red neuronal del párrafo anterior, llamada MLP (Perceptrón Multicapa) en la resolución de problemas de recomendación y generación de precondicionadores.

- Las redes neuronales están formadas por capas de perceptrones, éstas se entrenan utilizando datos de entrada y salida relacionados con sistemas de ecuaciones lineales y sus precondicionadores. Dichas redes tienen la capacidad de clasificar si es recomendable aplicar el método Simplex a un sistema dado y generar precondicionadores (ajustes dados ciertos criterios) óptimos para mejorar el rendimiento de dicho método.

A continuación se tiene una imagen que describe la estructura de una neurona y más específicamente un perceptron, que tiene los parámetro de entrada, función de activación que actúa según una distribución específica y la salida. Para el caso del anteproyecto si es conveniente o no l asolución o así mismo si el problema según el modelo posée solución.

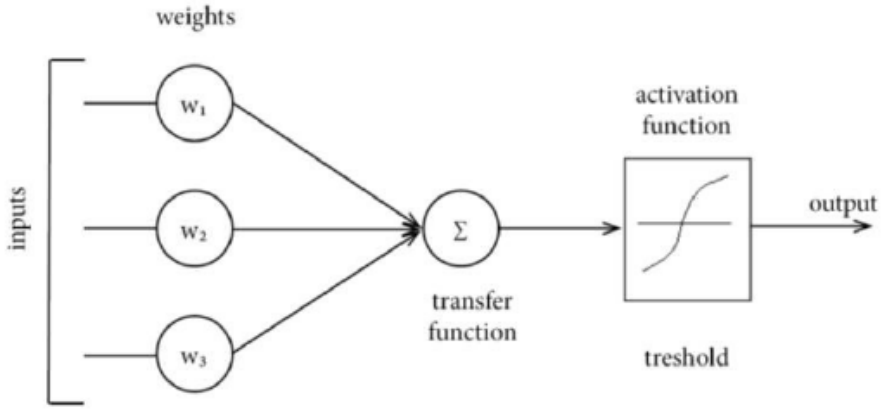

Figura 1: Perceptrón.

Uso de solver Ibex

En el documento se menciona la herramienta Ibex, la cual es una librería en C++ utilizada para resolver problemas de optimización global con restricciones no lineales. Ibex proporciona una plataforma flexible y eficiente para abordar una amplia gama de problemas de optimización.

En el contexto del anteproyecto, Ibex se utiliza como una referencia de validación para evaluar los resultados obtenidos mediante el uso de redes neuronales en la optimización de problemas complejos. Se emplea para realizar pruebas de rendimiento utilizando problemas no lineales almacenados en Ibex y en repositorios en línea. Las métricas de rendimiento, como el número de nodos, los tiempos de ejecución y los tiempos de Simplex, se comparan con los resultados obtenidos por las redes neuronales.

Figura 2: Libreía Ibex C++.

En resumen, Ibex desempeña un papel importante en el documento, ya que se utiliza como una herramienta de optimización global para comparar y validar los resultados obtenidos mediante el uso de redes neuronales en problemas complejos. Sirve como una referencia confiable y estable para evaluar el rendimiento y la eficacia de las soluciones propuestas.

Conclusiones finales del anteproyecto y sus resultados

En conclusión, el presente documento propone un enfoque basado en el uso de redes neuronales, especialmente redes MLP, para mejorar la eficiencia del método Simplex en problemas de optimización global. Se han explorado y comparado diferentes técnicas, como el uso de precondicionadores y la predicción del tiempo de ejecución, utilizando redes neuronales para recomendación y generación de precondicionadores adicionales. Estas técnicas han sido validadas y evaluadas en conjunto con el solver Ibex, una herramienta de optimización global de referencia.

Los resultados obtenidos han demostrado que el uso de redes neuronales puede mejorar significativamente la eficiencia del método Simplex en la resolución de problemas complejos de optimización global. Las redes neuronales MLP han mostrado su capacidad para estimar la conveniencia de aplicar el método Simplex y generar precondicionadores óptimos, lo que ha llevado a una reducción en el tiempo de ejecución y a una mejora en la calidad de las soluciones obtenidas.

En resumen, este estudio ha confirmado que el uso de redes neuronales en conjunción con el método Simplex puede ser una estrategia efectiva para abordar problemas de optimización global. Estas técnicas ofrecen una alternativa prometedora para optimizar el rendimiento del método Simplex en términos de eficiencia y calidad de las soluciones. Con futuras investigaciones y refinamientos, se espera que estas técnicas puedan ser aplicadas en una amplia gama de problemas de optimización global, brindando beneficios significativos en términos de tiempo de ejecución y precisión de los resultados.

3.2. Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects por Ignacio Araya y Victor Reyes

En este artículo se presenta una revisión detallada de los algoritmos de ramificación y acotación (Branch-and-Bound) para la optimización y la satisfacción de restricciones en intervalos. Los autores discuten los enfoques existentes, incluyendo la ramificación, acotación intervalar, acotación mixta y acotación híbrida. Además, el artículo describe los métodos basados en intervalos que son utilizados en diversas áreas de investigación para lidiar con incertidumbres en los valores de los parámetros y errores de cálculo. Estos métodos son capaces de tratar todo el espacio de búsqueda, lo que permite ofrecer pruebas de inviabilidad o certificación de soluciones.

Los métodos Interval Branch and Bound (B&B) son utilizados para resolver sistemas de restricciones continuas y problemas de optimización global. Aunque dichos métodos son matemáticamente rigurosos, en general, son menos eficientes que los optimizadores no lineales de la comunidad de programación matemática. Además, los autores también exploran las perspectivas futuras y las áreas de investigación abiertas en este campo, así como una revisión de los componentes principales de los algoritmos de intervalo B&B y ofrecen algunas consideraciones para mejorar el rendimiento.

Como objetivo principal de este documento se encontraron 2 solucionadores genéricos principales para resolver problemas de satisfacción de restricciones y optimización global; dichos solucionadores tienen componentes importantes en común: contractors y bisectores.

¿Qué son los Contractors?

Son un componente común en los solucionadores de problemas restricciones, que intentan eliminar valores de los dominios de las variables que no satisfacen una o más restricciones del sistema. Hay tres tipos de algoritmos contractors importantes que investiga y evalúa el documento, los de análisis de intervalos, los que utilizan CP (Constraint Problems o Problemas asociados a restricciones) y los que se basan en la relajación lineal.

Métodos contractores abordados

El método más efectivo de análisis de intervalos es la extensión del **método de Newton** a intervalos, que busca sucesivamente mejores aproximaciones a las soluciones de una función. El método de Newton se generaliza a intervalos y se aplica sucesivamente un paso de contrato para reducir el intervalo. Si el procedimiento converge a un punto fijo, devuelve un intervalo atómico que contiene la única solución de la ecuación en el intervalo. Si una iteración del paso de contrato devuelve un intervalo vacío, entonces no hay solución en el intervalo. El método se generaliza a n variables y n ecuaciones

usando el teorema del valor medio. Los cálculos se realizan en un cuadro de intervalos que contiene todas las soluciones. El principal inconveniente del método es que solo funciona con restricciones de igualdad y generalmente solo converge cuando los cuadros son lo suficientemente pequeños.

Los algoritmos basados en **programación de restricciones**, por otro lado, tratan con restricciones de desigualdad. Los solucionadores de restricciones utilizan diferentes técnicas para reducir el espacio de búsqueda y encontrar una solución factible. La programación de restricciones se puede aplicar a una amplia gama de problemas de optimización y satisfacción de restricciones.

Finalmente, los algoritmos basados en la **relajación lineal** son utilizados para encontrar soluciones aproximadas de problemas de optimización lineal. Estos algoritmos relajan las restricciones del problema original y resuelven el problema resultante en un espacio de menor dimensión. La solución obtenida en el espacio relajado se utiliza para producir una solución aproximada para el problema original. Los algoritmos basados en la relajación lineal son particularmente efectivos en problemas de gran escala y son utilizados en muchas aplicaciones prácticas.

- Los Contractors basados en Newton son útiles cuando el sistema está bien restringido y los espacios de búsqueda son lo suficientemente pequeñas.
- Los Contractors basados en relajación, son útiles para sistemas menos restringidos. En particular, los Contractors basados en relajación son contratistas lineales que relajan las restricciones no lineales del problema original a restricciones lineales. Esta técnica permite trabajar con sistemas de mayor tamaño y complejidad en comparación con otros tipos de contratistas.
- Los Contractors CP (programación de restricciones): Son eficientes para filtrar espacios de búsqueda más grandes, trabajando directamente en el sistema original de restricciones no lineales. Sin embargo, tienen un alcance reducido y sufren del problema de localidad.

¿Qué son los Bisectores?

Los bisectores son un componente importante de los algoritmos de intervalos B&B descritos en el paper. Estos métodos se utilizan para dividir los intervalos de búsqueda en dos subintervalos, de modo que se pueda enfocar la búsqueda en una región más específica que contiene la solución deseada. Los bisectores son particularmente importantes en la etapa de "branching" (ramificación) del algoritmo, en la cual se selecciona un subintervalo para explorar en detalle mientras se descartan otros subintervalos. El paper señala que solo unos pocos métodos de bisectores han sido propuestos hasta la fecha, y que los métodos basados en la técnica "SmearSumRel" parecen ser los más prometedores.

Métodos bisectores abordados

El método **Round Robin** es uno de los métodos bisectores de selección de variables más simples y utilizados en la optimización global. Este método no requiere información previa sobre el sistema y funciona de la siguiente manera: si el espacio de búsqueda actual de k-dimensiones se obtuvo dividiendo el intervalo $[\boldsymbol{x_i}]$, entonces el método round robin seleccionará la variable $\boldsymbol{x_j}$, con $\boldsymbol{j = (i + 1)\,\%k}$. El objetivo de este método es evitar descuidar cualquier variable.

Aunque es simple y fácil de implementar, una desventaja del método round robin es que un ordenamiento inicial de las variables puede afectar significativamente el desempeño del método. Es decir, si se selecciona una mala ordenación de las variables, el método puede tener un desempeño desastroso.

El método **Largest - First** es un algorítmo de selección de variable utilizado en métodos bisectores para la resolución de problemas de optimización global. El objetivo de este algoritmo es seleccionar la variable con el dominio más grande (intervalo más grande) para dividir el espacio de búsqueda actual en dos subespacios.

El algoritmo se basa en la suposición de que los intervalos con diámetros grandes tienen una mayor influencia en la evaluación de la función. Por lo tanto, seleccionar la variable con el dominio más grande maximiza la probabilidad de encontrar una solución óptima.

El algoritmo de "Largest-First" es simple y fácil de implementar. Sin embargo, tiene una desventaja de que puede descuidar algunas variables si no se ordenan adecuadamente en la selección inicial de variables. Además, este algoritmo no sigue el principio "fail-first"(fracasar primero), que sugiere seleccionar la variable con la probabilidad más alta de fallar para minimizar la longitud de la rama esperada y, por lo tanto, minimizar el costo de la búsqueda.

Algoritmo bisector "**Smear**" es un método de selección de variables utilizado en el contexto de problemas de optimización no convexa. Éste método se basa en la utilización del parámetro "smear value" (valor de difumación), que es una medida de la influencia de una variable sobre una función objetivo.

El valor de difuminación se calcula utilizando información del sistema y la variable con la mayor influencia se selecciona mediante la maximización de una agregación de los valores de difuminación en todo el sistema. La variante propuesta en el paper utiliza un valor de difuminación relativo en lugar del valor de difuminación absoluto, lo que lo hace más robusto según los experimentos realizados.

En resumen, el algoritmo bisector Smear es un método de selección de variables que utiliza la smear value como medida de la influencia de una variable y maximiza su valor agregado en todo el sistema para seleccionar la variable más influyente.

Conclusiones finales de la investigación

Los algoritmos de intervalos B&B se utilizan comúnmente para resolver problemas de satisfacción de restricciones y de optimización global. En este artículo, se describen dos solucionadores genéricos de intervalos B&B para tratar estos tipos de problemas. Tienen componentes importantes en común: contratistas y bisectores.

Los contratistas basados en Newton parecen ser una buena opción cuando el sistema está bien restringido y los espacios de búsqueda son lo suficientemente pequeños. Además, son útiles para certificar la existencia de una solución única en el espacio de búsqueda o caja. Los contratistas basados en relajación no se limitan a sistemas bien restringidos, pero las relajaciones lineales pueden perder información importante relacionada con el problema original. Los contratistas de CP son eficientes para filtrar cajas más grandes, trabajando directamente en el sistema original (no lineal) de restricciones. Sin embargo, tienen una vista reducida de todo el sistema y, por lo tanto, sufren del problema de la localidad. Las bibliotecas más recientes basadas en intervalos (por ejemplo, Ibex, Icos) permiten combinar estos tres enfoques en una estrategia de solucionador, lo que hace que los resultados sean más robustos.

En cuanto a los métodos de bisectación, hasta ahora solo se han propuesto algunos trabajos, siendo los métodos basados en difuminado (en particular, la heurística SmearSumRel) los más prometedores.

El problema de la dependencia ha sido un obstáculo para los solucionadores basados en intervalos desde el principio. Aunque aún queda trabajo por hacer, pensamos que los contratistas de CP sofisticados recientes abordan el problema de manera satisfactoria. Sin embargo, hemos observado varios problemas adicionales que deben tenerse en cuenta para mejorar el rendimiento del solucionador a corto plazo:

- Tratar el problema de la localidad con técnicas orientadas a la propagación de restricciones, como la combinación lineal de restricciones y la eliminación de subexpresiones comunes.
- Incorporar técnicas de contracción exitosas de algoritmos no rigurosos, como la técnica de reformulación de linealización, la reducción basada en la optimización, la convexificación y el uso de restricciones relacionadas con las condiciones de optimalidad orientadas a reducir el efecto de agrupamiento. La mayoría de estas técnicas deben adaptarse a los intervalos haciéndolos matemáticamente rigurosos.
- Incorporar mecanismos adaptativos para seleccionar contratistas. La idea aquí es aumentar el esfuerzo de contracción cuando es más probable filtrar dominios.
- Incorporar estrategias de retroceso para mejorar las heurísticas de selección de variables.
- Mejorar las técnicas de límite superior en problemas de optimización mediante la incorporación de algoritmos de búsqueda local más sofisticados (y posiblemente costosos). Para minimizar los costos, también deben incorporar mecanismos para controlar el esfuerzo.
- Mejorar la selección de la próxima caja en problemas de optimización. Por ejemplo, combinando criterios de intensificación y diversificación en una estrategia similar a KBFS.

3.3. The Simplex-Simulated Anneling Approach To Continous Non-Linear Optimization

Éste artículo presenta un algoritmo para la optimización global de funciona no convexas continuas, tanto sin restricciones como con restricciones. El algoritmo combina los métodos de simplex no lineal y "simulated annealing" basándose en una propuesta realizada por Press y Teukolsky. También se introduce una versión mejorada del algoritmo, en la cual el enfriamiento se aplica tan pronto como se encuentra una solución mejorada. Esta variante demuestra tiempos de ejecución más rápidos sin comprometer la calidad de las soluciones obtenidas.

Se realiza una prueba del algoritmo y su variante mejorada en varias funciones difíciles de la literatura (artículos científicos aledaños y correlacionados). Los resultados se comparan con los obtenidos utilizando un método de búsqueda aleatoria adaptativa robusta y el método simplex de Nelder y Mead. El enfoque propuesto resulta ser más robusto y eficiente para superar dificultades asociadas con óptimos locales, el vector de solución inicial y la secuencia de números aleatorios

¿Qué es y se como aplica para este contexto Simulated Annealing?

Es una técnica de optimización global que se basa en un proceso de enfriamiento de un material sólido. Se utiliza para encontrar soluciones aproximadas a problemas de optimización, tanto en espacios discretos como continuos.

En el contexto de este texto, Simulated Annealing se aplica a la optimización de funciones combinatorias y continuas. La idea principal es generar una serie de soluciones aleatorias y realizar cambios en ellas de manera iterativa. Estos cambios pueden ser aceptados o rechazados en función de una parámetro de aceptación basado en la diferencia de costos entre la solución actual y la nueva solución propuesta.

La técnica de Simulated Annealing se inspira en el proceso de recocido (Annealing) de los materiales, donde se calienta y enfría un material para obtener su estructura de energía mínima. De manera similar, durante la optimización, se permite que el algoritmo se "escape" de los óptimos locales moviéndose hacia soluciones de mayor costo en etapas tempranas del proceso. A medida que avanza el algoritmo, la probabilidad de aceptar soluciones de mayor costo disminuye, lo que permite la convergencia hacia la solución óptima. Similar un tanto a Hill_Climbing y Tabu_Search.

Se puede decir que Simulated Annealing es una técnica de optimización que utiliza un enfoque estocástico y de búsqueda aleatoria para encontrar soluciones aproximadas a problemas de optimización. Se basa en la simulación del proceso de enfriamiento de un material sólido y se aplica tanto a problemas combinatorios como continuos.

Explicación del algoritmo Simplex propuesto por Press y Teukolsky (1991)

Es un método de optimización utilizado para encontrar el mínimo de una función en un espacio multidimensional. Este algoritmo se utiliza en combinación con Simulated Annealing en los algoritmos SIMPSA y NE-SIMPSA.

El algoritmo Simplex comienza con un conjunto de puntos en el espacio que forman un simplex (un polígono en 2D, un tetraedro en 3D, etc). Éstos puntos son los vértices del simplex y representan posibles soluciones.

En cada iteración del algoritmo, se evalúa el valor de la función objetivo en cada uno de los vértices del simplex. Luego, se selecciona el mejor vértice (el que tiene menor valor de la función) y se procede a realizar una serie de operaciones para actualizar el simplex.

Estas operaciones incluyen la reflexión, la expansión y la contracción del simplex.

- La **reflexión** implica reflejar el peor vértice a través del centroide del resto de los vértices y evaluar el valor de la función en ese punto. Si el valor de la función en el punto reflejado es mejor que el valor del segundo peor vértice, se procede a la expansión.
- La **expansión**, implica extender el simplex más allá del punto reflejado.
- Si el valor de la función en el punto reflejado no es tan bueno como el segundo peor vértice, se realiza una **contracción**, que implica acercar el simplex al mejor vértice.

Las operaciones se repiten hasta que se cumpla algún criterio de terminación, como alcanzar un número máximo de iteraciones o que el tamaño del simplex sea lo suficientemente pequeño.

En definitiva, el algoritmo simplex propuesto por Press y Teukolsky es un método de optimización que utiliza un conjunto de puntos para formar un simplex y busca mejorar la solución a través de operaciones de reflexión, expansión y contracción. Este algoritmo se combina con el recocido simulado en los algoritmos SIMPSA y NE-SIMPSA para lograr una optimización más eficiente.

Cabe hacer mención de que hubo una gran cantidad de algoritmo complementarios para el simplex aplicado sobre el Simplex y que además poseen aplicaciones según investigaciones citadas en el Paper y/o documento. A continuación se dará una breve descripción de los 3 más importantes.

1. SIMPSA (Adaptación del Método de Simplex):
    - Algoritmo basado en simplex para la optimización de problemas lineales.
    - Utiliza operaciones de reflexión, expansión y contracción para mejorar gradualmente las soluciones.
    - Incorpora el concepto de enfriamiento global para reducir la magnitud de los movimientos del simplex a medida que avanza la optimización.
    - Se implementó en FORTRAN 77 y se probó en funciones de optimización no restringidas y restringidas.
    - Demostró converger a soluciones óptimas en diversos problemas de prueba.
2. NE-SIMPSA (SIMPSA No Equilibrio):
    - Variante del algoritmo SIMPSA que se enfoca en la búsqueda de soluciones óptimas sin seguir un equilibrio termodinámico.
    - Realiza movimientos adaptativos del simplex mediante una combinación de operaciones de reflexión, expansión y contracción.
    - Utiliza el enfriamiento global para controlar la magnitud de los movimientos del simplex.
    - Se implementó en FORTRAN 77 y se probó en funciones de optimización no restringidas y restringidas.
    - Demostró converger a soluciones óptimas de manera eficiente y robusta, incluso en problemas altamente no convexos y con restricciones.
3. MSGA (Adaptación de Búsqueda Aleatoria Multidireccional):
    - Algoritmo de búsqueda aleatoria multidireccional adaptativa.
    - Se utiliza como comparación con los algoritmos SIMPSA y NE-SIMPSA en el estudio.
    - Proporciona un enfoque de búsqueda aleatoria en varias direcciones para la optimización.
    - Requiere un mayor número de evaluaciones de la función objetivo en comparación con los algoritmos basados en simplex.
    - Utilizado en el estudio para evaluar la eficiencia y el rendimiento de los algoritmos SIMPSA y NE-SIMPSA.

Los algoritmo descritos anteriormente poseen diferentes enfoques y estrategias para la optimización de problemas no lineales y los resultados experimentales muestran la efectividad de SIMPSA y NE-SIMPSA en la convergencia de soluciones óptimas en una variedad de funciones de la prueba.

A continuación se presentan tablas y gráficas con resultados que demuestran el mejor funcionamiento con respecto a distintos escenarios aplicados sobre los algoritmos recién descritos.

Starting point	MSGA N_{Succ}*	Simplex N_{Succ}	Simplex N_{fobj}†	SIMPSA N_{Succ}	SIMPSA N_{fobj}	NE–SIMPSA N_{Succ}	NE–SIMPSA N_{fobj}
$(1.04400, 2.48909, 1.78541)^T$	87	0	511	100	31950	95	16702
$(2.22288, -0.17259, 1.98899)^T$	71	0	624	100	31500	96	15742
$(1.57411, -1.79381, -1.75688)^T$	70	0	531	99	31588	87	15352
$(2.04195, -0.95967, -1.90526)^T$	78	4	577	99	31626	94	15575
$(-1.41862, -0.16050, 1.01260)^T$	76	32	943	98	29915	96	15714
$(1.40498, -0.24513, -1.00200)^T$	76	18	942	100	31195	92	15779
$(-1.3, -0.3, 0.63)^T$	75	29	958	96	30516	90	14610
Average	76	12		99		93	

* Percent number of successes.
† Average number of function evaluations.

Figura 3: Los resultados para la función restringida 6 ($\boldsymbol{\delta} = 10$) para SIMPSA y $\boldsymbol{\delta} = 10$ para NE-SIMPSA.

Function	Starting point	Simplex	SIMPSA	NE–SIMPSA
7	$(25, 1, 1)^T$	845	44413	23839
	$(30, 0, 0.6)^T$	631	44622	24651
8	$(1, 1, 1, 0.3, 0)^T$	954	59842	23970
	$(0, 0, 0.3, 1.2, 0)^T$	1546	46734	23214
9	$(28, 14, 14, 110, 110, 110, 110)^T$	2040	52402	13780
			298228*	60037*

* $\delta = 10^{-2}$ for SIMPSA and $\delta = 10^{-5}$ for NE–SIMPSA.

Figura 4: Número promedio de evaluación de la función objetivo (100 ejecuciones) para las funciones restringidas 7-9 ($\boldsymbol{\delta} = \mathbf{10^{-1}}$ para SIMPSA y $\boldsymbol{\delta} = \mathbf{10^{-4}}$ para NE-SIMPSA)

Observaciones de tablas (figura 3 y figura 4):

- De lo que se puede ver en los resultados visualizados en ambas tablas es que el algoritmo MSGA requiere de más evaluaciones de funciones que las implementaciones de SIMPSA y NE-SIMPSA y produce resultados de no mucho éxito en contraste SIMPSA y NE-SIMPSA. Los tiempos promedio de ejecución fueron de 0.28s, 1.00s, 1.20s y 1.70s, respectivamente, para los algoritmos simplex, NE-SIMPSA, MSGA y SIMPSA.
- Además el número de promedio de evaluaciones de la función objetivo requeridas por todos los algoritmos para resolver las funciones de 7 a 9 (ver figura 4). Dichos valores indican el esfuerzo de recursos computacionales para encontrar soluciones óptimas.

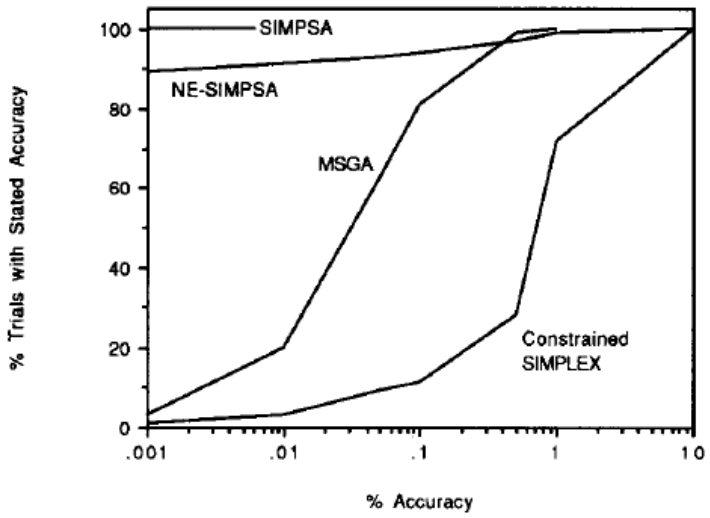

Figura 5: Número promedio de evaluación de la función objetivo (100 ejecuciones) para las funciones restringidas 7-9 ($\boldsymbol{\delta} = \mathbf{10^{-1}}$ para SIMPSA y $\boldsymbol{\delta} = \mathbf{10^{-4}}$ para NE-SIMPSA)

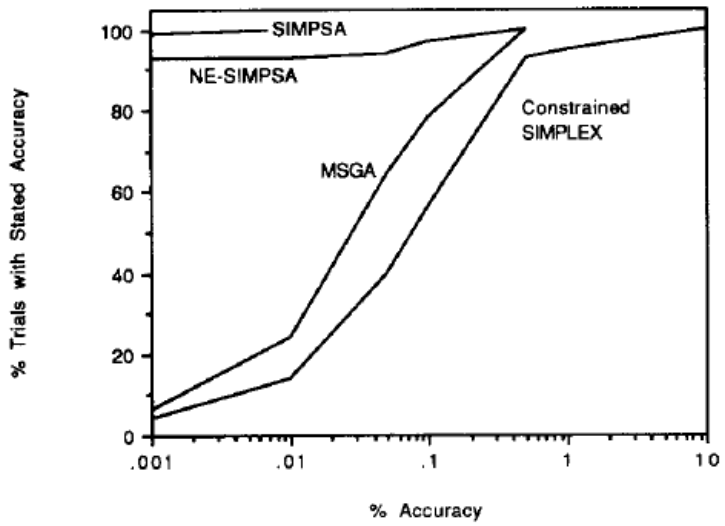

Figura 6: Número promedio de evaluación de la función objetivo (100 ejecuciones) para las funciones restringidas 7-9 ($\boldsymbol{\delta} = \mathbf{10^{-1}}$ para SIMPSA y $\boldsymbol{\delta} = \mathbf{10^{-4}}$ para NE-SIMPSA)

Observaciones de Gráficas (figura 5 y figura 6):

- Para el caso de la primera gráfica, el algoritmo simplex restringido es el menos preciso y robusto, ya que se encuentra atrapado en un óptimo local en 66 de las 100 ejecuciones. Los otros tres algoritmos (MSGA, NE-SIMPSA y SIMPSA) logran resultados dentro del 1 % del óptimo global. El algoritmo MSGA, aunque no es el más preciso, logra escapar en la mayoría de las ocasiones (92 de 100) del óptimo local, mientras que SIMPSA se atrapa en 14 ocasiones y NE-SIMPSA en 35 ocasiones. En este caso, NE-SIMPSA requiere la menor cantidad de evaluaciones de función.
- Para el caso de la segunda gráfica, se observa una clasificación similar entre los algoritmos, siendo el simplex restringido el de peor rendimiento y los otros tres algoritmos proporcionando resultados dentro del 1 % del óptimo global. Una vez más, NE-SIMPSA requiere la menor cantidad de evaluaciones de función entre los tres métodos robustos. Los tiempos promedio de ejecución fueron de 0.42 s, 2.08 s, 2.29 s y 4.20 s respectivamente para los algoritmos simplex restringido, MSGA, NE-SIMPSA y SIMPSA.
- Comportamiento de los algoritmos: Las Figuras 5 y 6 muestran el comportamiento de los algoritmos en la optimización del problema de asignación de combustible (función 7) para dos puntos de partida diferentes. Estas figuras

muestran el número de intentos que alcanzaron un valor de la función objetivo dentro de un cierto porcentaje del óptimo global.

Estos resultados sugieren que no se debe esperar llegar al óptimo global con una sola ejecución del algoritmo en una computadora, independientemente de la robustez esperada del algoritmo.

Obervaciones finales y conclusiones de la investigación del presente documento

1. El algoritmo SIMPSA (Simplex-Simulated Annealing) es más eficiente y robusto en comparación con el método simplex y los algoritmos NE-SIMPSA y MSGA para la optimización de funciones difíciles y restringidas.
2. El algoritmo NE-SIMPSA requiere menos evaluaciones de funciones que el algoritmo SIMPSA y generalmente menos que el algoritmo MSGA, lo que lo hace más eficiente en general.
3. En términos de tiempos de ejecución promedio, el orden de eficiencia es el siguiente: simplex restringido $<$ NE-SIMPSA $<$ MSGA $<$ SIMPSA. Con un enfriamiento más lento, los tiempos de ejecución aumentan significativamente para NE-SIMPSA y SIMPSA.
4. En general, los algoritmos propuestos SIMPSA y NE-SIMPSA son más robustos y eficientes, con la versión de no equilibrio (NE-SIMPSA) siendo más eficiente. Representan alternativas interesantes para la optimización global de problemas de interés para la ingeniería química y también para aplicar alguna variable de decisión a partir de heurística estocástica y/o determinista.
5. El método simplex implementado es mucho menos eficiente que los algoritmos SIMPSA, NE-SIMPSA y MSGA para la optimización de funciones restringidas utilizadas en el trabajo actual.
6. Se sugiere que futuras extensiones de este trabajo incluyan el desarrollo de un algoritmo eficiente de tipo SIMPSA para la optimización de problemas de programación no lineal mixta entera (MINLP) difíciles.

En resumen, los algoritmos SIMPSA y NE-SIMPSA son más eficientes y robustos en comparación con el método simplex y el algoritmo MSGA para la optimización de funciones difíciles y restringidas. Son alternativas interesantes para la optimización global de problemas.

3.4. Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks

El Paper se enfoca en seleccionar el algoritmo más eficiente para resolver problemas de programación lineal. Los investigadores utilizaron redes neuronales artificiales (ANN) para predecir el tiempo de ejecución de los algoritmos *simplex primal* y *simplex dual*, que son dos de los métodos más utilizados.

Cabe mencionar que el documento señala que en trabajos anteriores, investigadores y otros autores correlacionados habían desarrollado un modelo de predicción para el método del punto interior, pero en esta investigación, ampliaron su campo de observación para concluir los algoritmos simplex primal y dual.

Los resultados mostraron que no pudieron crear un modelo preciso para predecir el tiempo de ejecución exacto de estos algoritmos. En cambio, abordaron el problema como uno de clasificación, donde el modelo proporciona rangos de tiempo en los que se espera que se encuentre el tiempo de ejecución.

Los modelos de clasificación obtuvieron buenos resultados, con una precisión del 83 % para el algoritmo primal y del 84 % para el algoritmo dual.

En resumen, el estudio utiliza redes neuronales artificiales para predecir el rendimiento de los algoritmos simplex primal y dual en problemas de programación lineal. Aunque no pudieron predecir el tiempo exacto de ejecución, se logró clasificar el tiempo en rangos con una precisión satisfactoria.

Sección de regresión y Sección de clasificación

Su objetivo principal es encontrar modelos eficientes para predecir clases de instancias de lugar o espacios de búsqueda para valores exactos obtenidos luego de la ejecución de algún algoritmo.

1. En la sección de ***Regresión***, se comparan diferentes modelos de regresión como (Árbol de decisiones, ElasticNet, Lasso, Linear, Random Forest Ridge y Support Vector Machine) utilizando métricas como RMSE, MAE, MedAE y R2. Sin embargo, los modelos de regresión no se consideran seguros o válidos para la predicción precisa del tiempo de ejecución, por lo que se decide experimentar con técnicas de clasificación.

    - Los métodos de regresión se utilizaron con el objetivo de predecir el tiempo de ejecución del algoritmo simplex primal y del algoritmo simplex dual. Se realizaron pruebas exhaustivas con diferentes modelos de regresión a través de un conjunto de datos para aplicación de un entrenamiento y con ello se evaluó la efectividad con un conjunto de datos de prueba.
    - El objetivo era encontrar un modelo de regresión que pudiera predecir de manera precisa y precisa el tiempo de ejecución de los algoritmos simplex primal y dual. Sin embargo, después de evaluar los modelos de regresión, se decidió que los modelos de regresión no eran la mejor opción y se optó por utilizar técnicas de clasificación en su lugar.

2. Para los métodos de ***Clasificación***, se prueban modelos utilizando algoritmos como MLPClassifier y KNeighborsClassifier. Se explican métricas como precisión, recall, F1 y soporte, y se analizan las matrices de confusión y los informes de clasificación para evaluar el rendimiento de los modelos de clasificación generados. Se establecen cuatro clases para el tiempo de ejecución de los algoritmos simplex primal y dual, y se seleccionan los modelos más eficientes para cada algoritmo.

    - En el contexto de los algoritmos simplex primal y dual, las cuatro clases para el tiempo de ejecución pueden variar dependiendo de la definición específica utilizada en un estudio o implementación en partícular. Sin embargo, a menudo se definen las siguientes clases generales para el tiempo de ejecución:
        a) *Tiempo de ejecución óptimo*: Ésta clase se refiere a los casos en los que el algoritmo simplex primal o dual encuentra un a solución óptima en un tiempo razonable. En este caso, el algoritmo converge rápidamente y se considera "eficiente".
        b) *Tiempo de ejecución subóptimo*: En esta clase, el algoritmo simplex primal o dual encuentra una solución, pero no es óptima. Puede haber una brecha entre la solución encontrada y la solución óptima. Aunque el algoritmo no alcanza la solución óptima, el tiempo de ejecución sigue siendo razonable.
        c) *Tiempo de ejecución lento*: Esta clase se refiere a los casos en los que el algoritmo simplex primal o dual tarda un tiempo considerable en converger hacia una solución. Puede haber múltiples razones para un tiempo de ejecución lento, como una formulación del problema compleja, estructuras de datos ineficientes o características específicas del conjunto de datos.
        d) *Tiempo de ejecución no factible*: En esta clase, el algoritmo simplex primal o dual no puede encontrar una solución factible dentro de un límite de tiempo razonable. Esto puede deberse a problemas con la formulación del problema, la estructura de datos o las características del conjunto de datos que hacen que el algoritmo se atasque o se vuelva ineficiente.

¿Qué se puede decir acerca de los resultados obtenidos en la investigación?

1. Para modelos de regresión el paper utilizó el algoritmo MLPRegressor para desarrollar modelos de regresión para el tiempo de ejecución de los algoritmos simplex primal y dual. Se probaron diferentes configuraciones de parámetros, cómo el número de capas ocultas, la función de activación y el solucionador utilizado.

Algorithm	Primal, Dual
Hidden layers	1–3
Hidden layer sizes	10–100 neurons/layer
Activation function	relu, tanh, logistic
Solver	lbfgs, sgd
Alpha value	1×10^{-5}
Maximum iterations	1000
Tolerance	0.0001

Figura 7: MLPRegressor algoritmos en Simplex primal y dual.

2. En el caso de las métricas de evaluación, las tablas presentan diferentes métricas de evaluación utilizadas para medir rendimiento de los modelos de regresión. Éstas métricas incluyen el error cuadrático medio (RMSE), el error absoluto medio (MAE), el error absoluto mediano (MedAE) y el coeficiente de determinación. Éstas métricas proporcionan información sobre la precisión y la calidad de los modelos en la predicción del tiempo de ejecución.

	Primal		Dual	
	Training Set	Test Set	Training Set	Test Set
RMSE	342.08	1302.25	345.28	1260.39
MAE	9.60	25.07	11.35	25.16
MedAE	3.26	14.75	3.93	16.05
R^2	0.79	0.21	0.66	0.05

Figura 8: Modelo MLPRegressor para ejecución de tiempo de Simplex primal y Simplex dual.

3. Para la evaluación de modelos, los valores de las métricas de evaluación revelan que los modelos de regresión desarrollados no son adecuados para predecir el tiempo de ejecución de los algoritmos simplex primal y dual. Los valores de $\boldsymbol{R^2}$ son bajos y, en algunos casos, incluso negativos, lo que indica que los modelos no siguen la tendencia de los datos y no son útiles para fines de predicción.

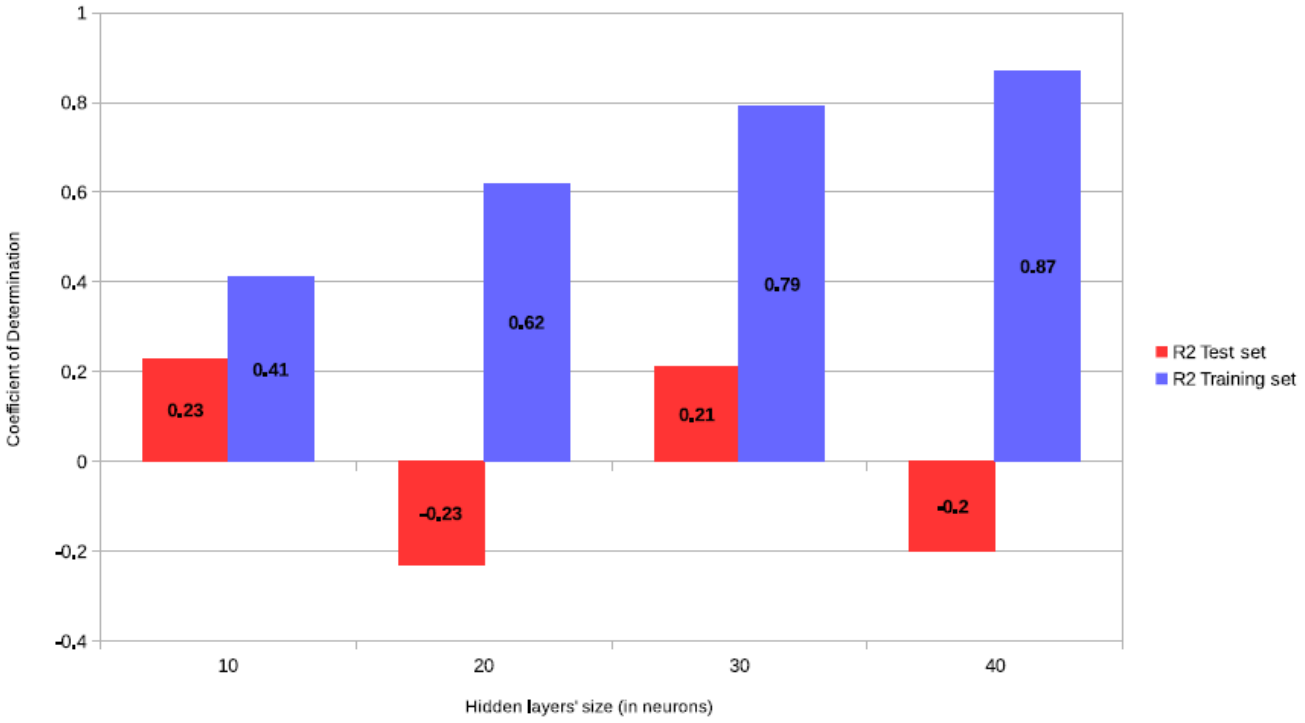

Figura 9: Modelo de regresión para el método primal: ajuste del número de neuronas en capas ocultas.

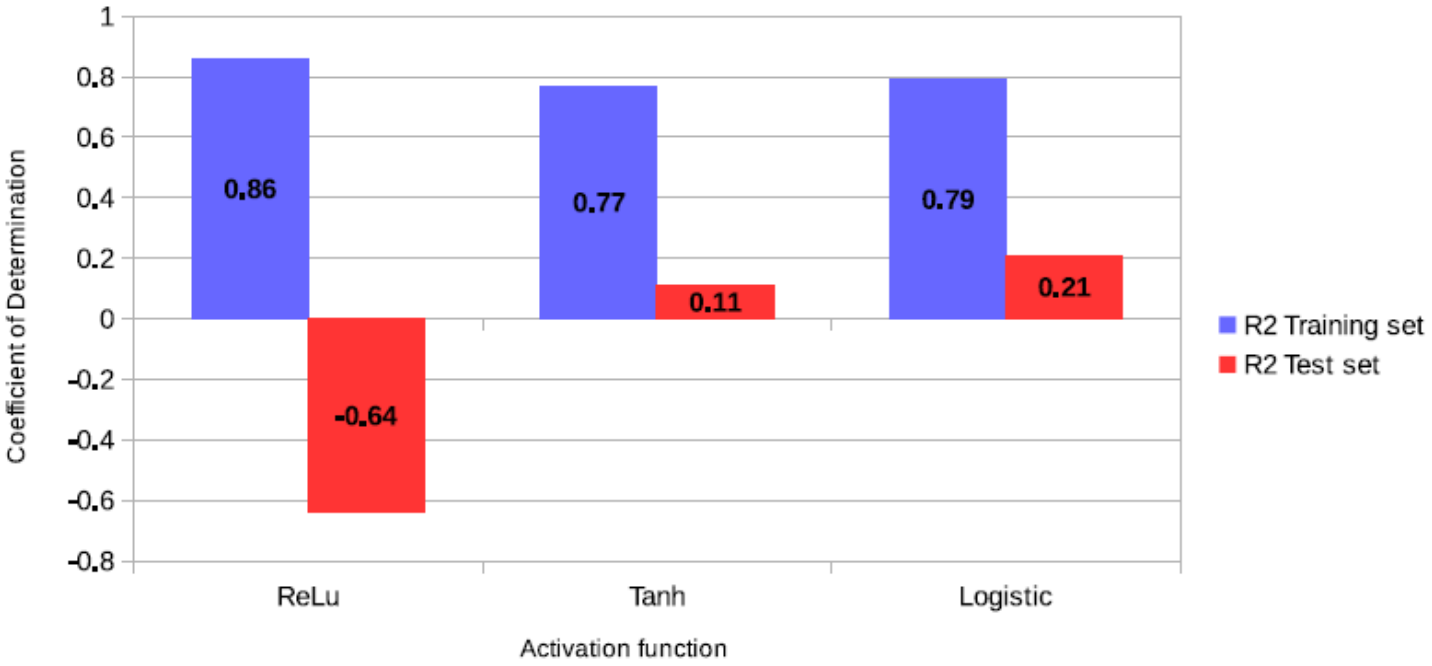

Figura 10: Modelo de regresión para el método primal: ajuste de la función de activación.

4. Respecto a la comparación de modelos, la investigación también compara los modelos de regresión desarrollados utilizando diferentes configuraciones de parámetros. Se muestran gráficas que representan los valores de $\boldsymbol{R^2}$ para diferentes números de neuronas en las capas ocultas, diferentes funciones de activación y diferentes solucionadores. Estas gráficas ilustran cómo varía el rendimiento de los modelos con respecto a las diferentes configuraciones probadas.

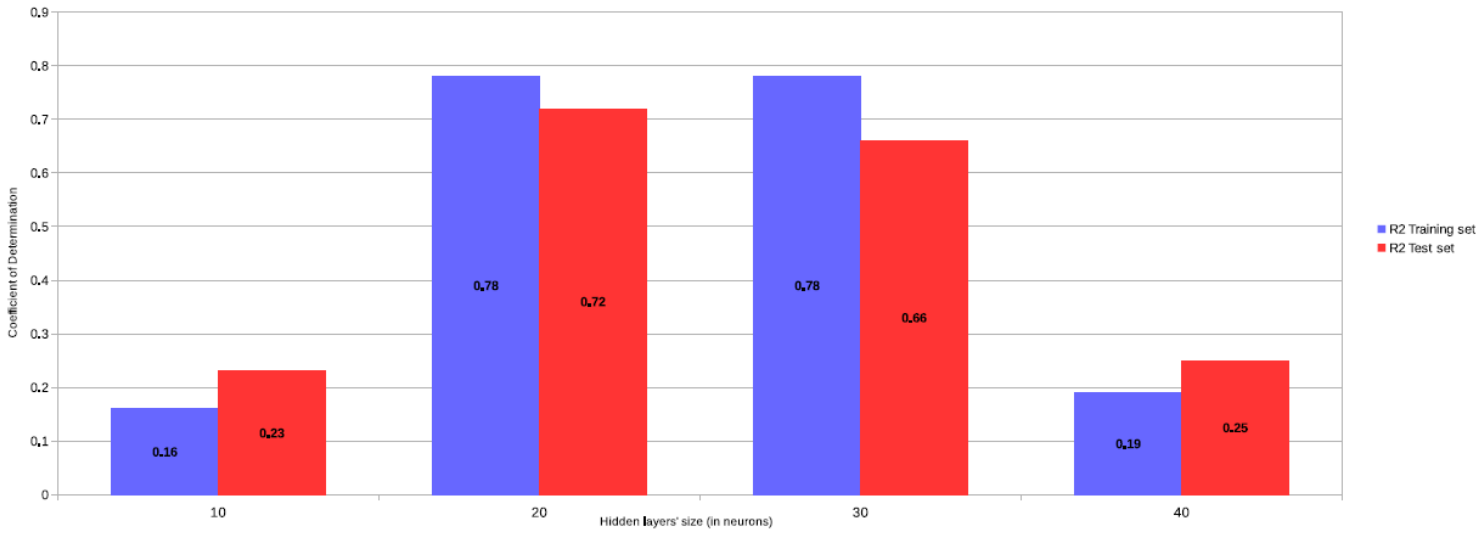

Figura 11: Modelo de regresión para el método de punto interior: ajuste del número de neuronas (1 capa oculta).

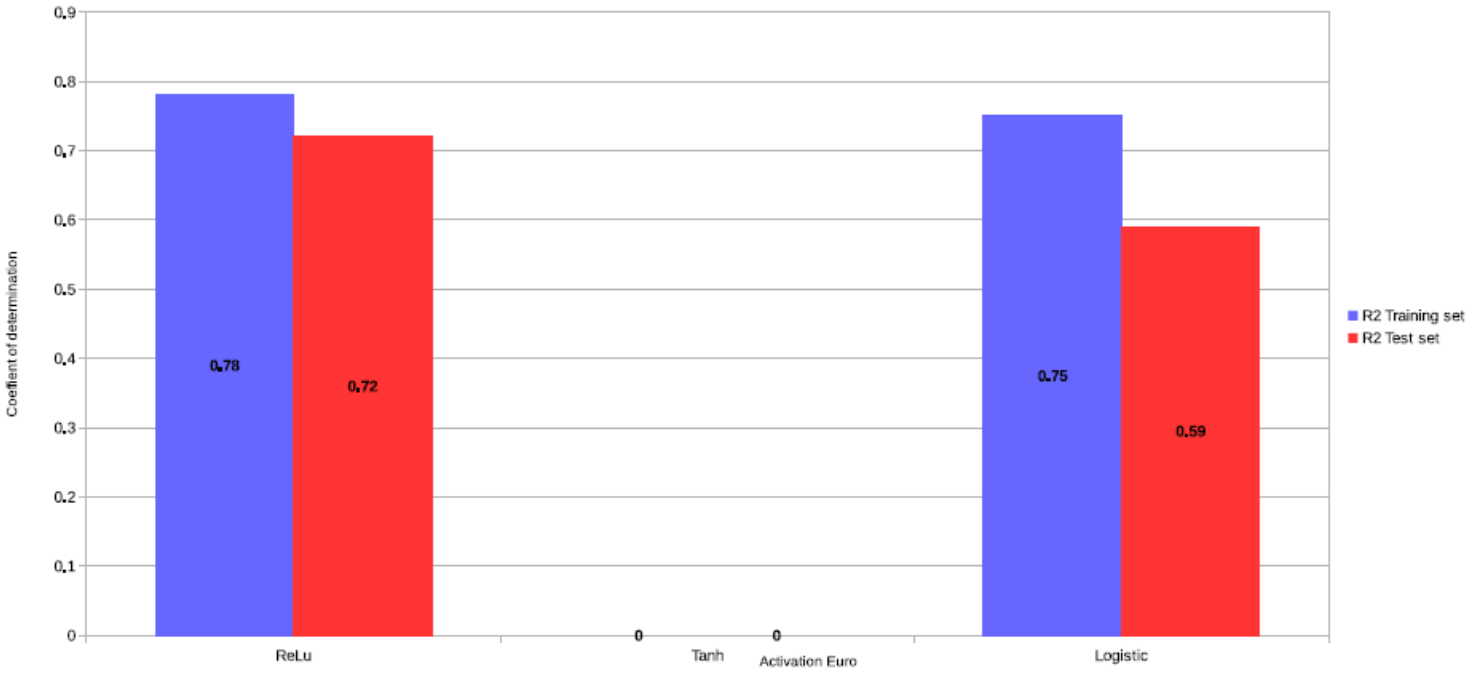

Figura 12: Modelo de regresión para el método de punto interior: ajuste de la función de activación.

5. Incluye también otras técnicas de regresión: Además de MLPRegressor, el paper menciona que se evaluaron otros algoritmos de regresión, como Decision Tree, ElasticNet, Lasso, Random Forest, Ridge, Support Vector y Linear Regressor. Sin embargo, se concluyó que ninguno de estos modelos era adecuado para predecir el tiempo de ejecución de los algoritmos simplex primal y dual.

	Training Set				Test Set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1441.9	31.77	26.02	0.01	2098.2	37.06	28.00	−0.03
ElasticNet	1483.2	32.11	24.28	0.07	1717.2	35.04	27.27	−0.01
Lasso	1470.6	32.24	35.39	0.05	1797.1	35.17	24.92	−0.002
Linear	1669.9	34.95	27.82	0.05	1177.7	29.90	24.39	−0.03
Random Forest	209.4	10.28	7.57	0.88	954.20	22.52	17.53	0.15
Ridge	1491.6	32.10	24.50	0.04	1809.10	36.41	28.00	−0.01
Support Vector	1663.0	27.55	17.89	−0.08	2138.28	31.92	18.46	−0.14

Figura 13: Modelos de regresión sobre Simplex primal, respecto a ejecuciones de tiempo y un delta de tiempo en Segundos.

	Training Set				Test Set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1125.2	28.02	23.90	0.02	998.6	26.98	25.90	−0.08
ElasticNet	1014.7	26.16	20.90	0.03	1206.8	27.90	20.76	0.04
Lasso	940.2	24.01	18.77	0.09	1327.0	28.93	20.92	−0.08
Linear	1040.6	25.87	17.59	0.10	1006.7	25.99	22.91	−0.10
Random Forest	148.4	8.68	6.77	0.87	618.9	17.29	13.12	0.34
Ridge	1033.3	26.21	19.78	0.11	997.9	24.84	18.60	−0.10
Support Vector	1299.2	21.26	8.06	−0.17	1356.96	24.15	12.20	−0.30

Figura 14: Modelos de regresión sobre Simplex dual, respecto a ejecuciones de tiempo y un delta de tiempo en Segundos.

En resumen, las tablas y gráficas del paper revelan que los modelos de regresión desarrollados no son efectivos para predecir el tiempo de ejecución de los algoritmos simplex, primal y dual. Se realizó un análisis exhaustivo utilizando diferentes configuraciones de parámetros y algoritmos de regresión, pero los resultados mostraron un bajo rendimiento en términos de precisión y predicción.

3.5. Nonlinear Models and Problems in Applied Sciences from Differential Quadrature to Generalized Collocation Methods

El artículo trata sobre el método de cuadratura diferencial y su aplicación en la solución de problemas en ciencias aplicadas. El método de cuadratura diferencial es una técnica que utiliza interpolación para aproximar soluciones de ecuaciones diferenciales.

El paper discute los fundamentos teóricos del método y cómo se puede aplicar a problemas con condiciones de contorno no lineales. También explora mejoras en las técnicas de interpolación, sugiriendo el uso de aproximaciones espectrales en lugar de interpolaciones tradicionales.

Se presentan resultados de experimentos computacionales y se discuten las consideraciones para seleccionar los parámetros óptimos, como el número de puntos de colocación y el paso de integración en el tiempo.

En resumen, el paper ofrece una visión general del método de cuadratura diferencial, su aplicabilidad en problemas de ciencias aplicadas y propone perspectivas de investigación para su desarrollo y mejora.

¿Qué se puede decir acerca de los resultados obtenidos en la investigación?

Basándose en la información y en las actividades realizadas por la investigación se pude decir que:

1. El método de cuadratura diferencia es una técnica eficiente para resolver una variedad de problemas en ciencias aplicadas. Ha sido propuesto y desarrollado por varios autores a lo largo del tiempo.

2. El método puede manejar problemas con condiciones de contorno no lineales, que son más realistas en comparación con las condiciones lineales de Dirichlet o Neumann. Las condiciones de contorno pueden incluir condiciones de compatibilidad no lineales en el límite del dominio del problema

3. El método se basa en el uso de interpolantes fundamentales, como polinomios de Lagrange o funciones seno. Se sugiere que las funciones seno son más adecuadas para soluciones oscilantes en el espacio, mientras que las interpolaciones de Lagrange son más apropiadas para soluciones no oscilantes.

4. Se propone investigar y desarrollar técnicas de interpolación mejoradas, como aproximaciones espectrales, que pueden proporcionar una representación más precisa de la función y mejorar la precisión de los resultados.

5. En términos de la selección de parámetros, se sugiere realizar análisis experimental y computacional para determinar los valores óptimos de los pasos de integración en el tiempo y el número de puntos de colocación. Esto puede ayudar a mejorar la precisión de las soluciones obtenidas.

4. Metodología propuesta

Basándonos en la investigación realizada y los resultados obtenidos, podemos definir una metodología para mejorar la aplicación del algoritmo simplex en problemas y parámetros de entrada utilizando una red MLP (Perceptrón Multicapa). Aquí hay una propuesta de cómo podría estructurarse esa metodología:

1. Recopilación y preparación de datos: El proceso de recopilación y preparación de datos se mantiene igual. Asegúrate de tener conjuntos de datos que contengan información sobre problemas de programación lineal y sus características, junto con una etiqueta que indique si es conveniente o no aplicar la solución.

2. Diseño y entrenamiento de la red neuronal clasificadora: Utiliza una red neuronal adecuada para clasificación binaria, como una red MLP con una capa de salida de un solo nodo utilizando una función de activación sigmoidal. Diseña y entrena la red utilizando los conjuntos de entrenamiento y prueba disponibles.

3. Validación cruzada y ajuste de hiperparámetros: Aplica técnicas de validación cruzada para evaluar el rendimiento de tu red clasificadora en diferentes conjuntos de prueba. Ajusta los hiperparámetros de la red, como la tasa de aprendizaje, el número de capas y neuronas, utilizando técnicas de optimización como la búsqueda en cuadrícula o la optimización bayesiana.

4. Evaluación del rendimiento y comparación con el enfoque sin la red neuronal: Evalúa el rendimiento de la red clasificadora en términos de precisión en la predicción de si es conveniente o no aplicar la solución. Compara este enfoque con el enfoque tradicional sin la red neuronal para determinar si la incorporación de la red mejora el rendimiento y la eficiencia del proceso de toma de decisiones.

5. Optimización de los vértices del politopo: En este caso, la optimización de los vértices del politopo no se aplica, ya que la red neuronal se enfoca únicamente en decidir si es conveniente o no sacar la solución. Sin embargo, si deseas utilizar técnicas de optimización para mejorar el proceso de búsqueda, puedes explorar métodos como el algoritmo genético o la optimización basada en enjambres (por ejemplo, el algoritmo de optimización por enjambre de partículas - PSO) para encontrar configuraciones óptimas o cercanas a la óptima.

Representación esquemática del algoritmo:

1. Recopilación y preparación de datos:

   - Obtener conjuntos de datos de problemas de programación lineal y sus características.
   - Asignar etiquetas a cada problema indicando si es conveniente o no aplicar la solución.
   - Dividir los datos en conjuntos de entrenamiento y prueba.

2. Diseño y entrenamiento de la red neuronal clasificadora:

   - Inicializar la estructura de la red neuronal, utilizando una red MLP con una capa de salida de un solo nodo con función de activación sigmoidal.
   - Definir los hiperparámetros de la red, como la tasa de aprendizaje, el número de capas y neuronas.

- Iterar a través de los datos de entrenamiento:
  - Propagar hacia adelante los datos de entrada a través de la red neuronal.
  - Calcular la función de pérdida utilizando la salida de la red y las etiquetas verdaderas.
  - Propagar hacia atrás el error y ajustar los pesos de la red utilizando un algoritmo de optimización como el descenso de gradiente.
- Repetir el paso anterior hasta que la red converja o se alcance un número máximo de iteraciones.

3. Validación cruzada y ajuste de hiperparámetros:
   - Dividir los datos de entrenamiento en K pliegues para realizar validación cruzada.
   - Para cada combinación de hiperparámetros:
     - Iterar sobre los pliegues de validación cruzada:
       - Utilizar el conjunto de entrenamiento actual para entrenar la red neuronal.
       - Utilizar el conjunto de validación para evaluar el rendimiento de la red con los hiperparámetros actuales.
     - Calcular la medida de rendimiento promedio para cada combinación de hiperparámetros.
   - Seleccionar la combinación de hiperparámetros con el mejor rendimiento.
4. Evaluación del rendimiento:
   - Utilizar el conjunto de prueba para evaluar el rendimiento final de la red neuronal.
   - Comparar el rendimiento con el enfoque tradicional sin la red neuronal.
5. Optimización de los vértices del politopo (opcional):
   - Aplicar un algoritmo de optimización (como el algoritmo genético o PSO) para encontrar configuraciones óptimas o cercanas a la óptima de los vértices del politopo.
6. Resultados y conclusiones:
   - Analizar los resultados obtenidos y evaluar el rendimiento y la eficiencia de la red clasificadora.
   - Determinar si la incorporación de la red neuronal mejora el proceso de toma de decisiones en comparación con el enfoque tradicional.
   - Extraer conclusiones y realizar recomendaciones para futuras mejoras.

5. Problema abordado

Para abordar la problemática se tuvo que considerar el cambio de precondicionadores en donde se evaluó como la red neuronal cambiaba según nuevas métricas y considerando un modelo en específico compilado por la librería keras.

Las métricas de entrada vienen a ser las siguientes:

- La probabilidad juega un papel crucial en la tendencia de convergencia y aprendizaje. Su objetivo principal es estimar un porcentaje para restringir la solución y, al mismo tiempo, proporcionar un margen para el recorte de los vértices del dominio del politopo. Esto resulta muy útil al construir la "Matriz P", que es una matriz de precondicionadores creada después de la linealización de Taylor. La linealización de "Taylor" es de suma importancia para comprender y conceptualizar la transformación de un problema no lineal a uno lineal, el cual puede ser evaluado como un sistema de ecuaciones con una cantidad de salida de "n" variables.
- Número de variables de entrada: para el caso del aprendizaje aunque no sea menor es de suma importancia identificar que la precisión debe de ser calculada por sobre la cantidad de variables de entrada y de salida ya que no es solamente una variable que se evalúa sino también un número concreto de variables en un sistemas de ecuaciones que componen a una o más expresiones.
- La precisión se basa en las variables de entrada y las variables de salida. Es crucial tener en cuenta cómo varía el coeficiente de aprendizaje de la precisión en función de la cantidad de variables tanto de entrada como de salida. Además, es importante estimar la precisión considerando cada vértice del problema, ya que la métrica para determinar el vértice óptimo debe incluir los mejores vértices dentro de subconjuntos de tamaños similares o cercanos al dominio inicial y de salida.

¿Cómo se evaluarán dichos cambios?

Se utilizará una red neuronal con un modelo de aprendizaje para realizar mejoras en el problema. Se realizarán 5 intentos fijos y se analizarán variables como el número de nodos recorridos en el problema. Además, se evaluarán problemas específicos categorizados como Ibex (Easy, Medium y Hard). Los valores se registrarán en un archivo con extensión .log y luego serán procesados por un productor y un consumidor.

El productor contendrá toda la información del contenedor de Ibex, mientras que el consumidor se encargará del procesamiento de los datos enviados por Ibex. Utilizando bibliotecas como matplotlib en Python, se generarán gráficos y se evaluarán los datos obtenidos durante el procesamiento. También se realizará un breve análisis adjunto a cada experiencia abordada.

Nota: La razón por la que se está utilizando un par de productor-consumidor es debido a ciertas limitaciones encontradas en el programa desarrollado por el memorista Nicolás Calderón. En el programa original, la información no se enviaba correctamente de un proceso a otro, lo cual podría ser problemático, especialmente si Ibex experimenta algún problema y la información se pierde. Para evitar esta situación, se decidió crear archivos de seguimiento y enviar el contenido de dichos archivos de un software a otro. Además, el volcado de archivos previos podría afectar el rendimiento del programa, lo cual podría generar deficiencias en ciertos procesos ejecutados por Ibex o incluso generar incongruencias; por lo que de forma no muy minuciosa se hicieron pruebas de funcionamiento dando resultados favorables y así mismo nuevos.

Diagrama de ambiente para desarrollo de solución

A continuación se presenta una arquitectura para el desarrollo y donde se ubica el ambiente pruebas:

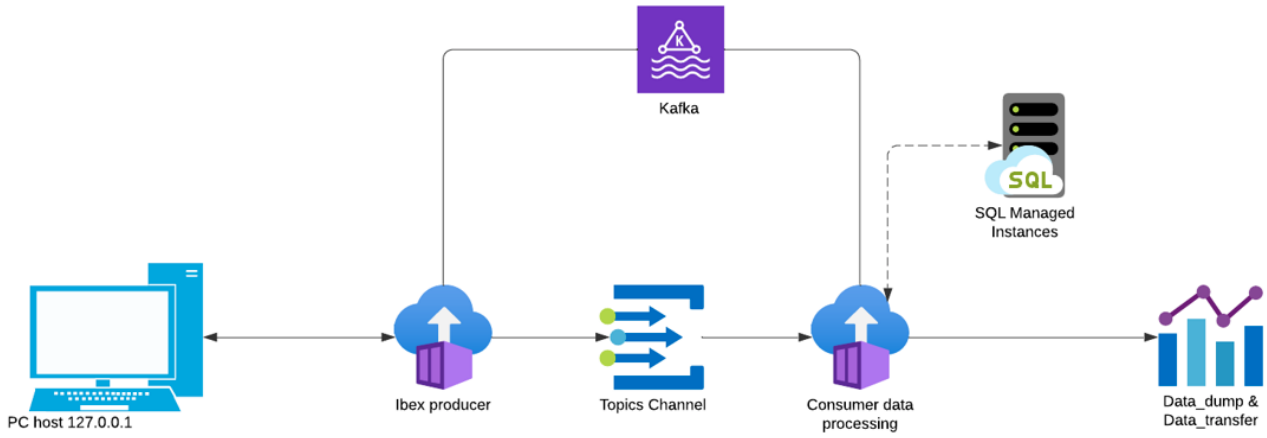

Figura 15: Diagrama del ambiente pruebas.

Descripción de ambiente:

- PC host: el primer nodo se encarga de virtualizar el ambiente en donde se hostería a través de una gran red los contenedores tanto de ibex y del procesamiento de datos. La red principal vendría a tener el nombre del fichero en el cual se encuentra el "docker-compose.yaml" (dicho archivo contiene toda la arquitectura del sistema).
- Ibex producer: es un contenedor con una imagen de Ubuntu con la versión 22:04 para ser más precisos, con todas las dependencias de ibex alojadas en esta en donde también se vincula con kafka para el enviado de los datos posterior a la ejecución de problemas de optimización en ibex. Para el compilado se deben de seguir ciertas isntrucciones:
  - Primero, se debe de hacer un configure de todos las dependencias de los archivos de ibex.

```
root@50a963765986:/app# ./waf configure --interval-lib=gaol --lp-lib=so
plex
Setting top to                           : /app
Setting out to                           : /app/__build__
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for program 'bison'             : /usr/bin/bison
Checking for program 'flex'              : /usr/bin/flex
sys.platform                             : linux
Checking for compiler flags -std=c++11   : yes
Checking for compiler flags -O3          : yes
Checking for compiler flags -Wno-deprecated : yes
Checking for compiler flags -Wno-unknown-pragmas : yes
Checking for compiler flags -Wno-unused-variable : yes
Checking for compiler flags -Wno-unused-function : yes
Checking for compiler flags -U__STRICT_ANSI__    : yes
Ibex will be built as a                  : static library
Configuration of the library for interval arithmetic
```

Figura 16: Ejecución de comando configure.

- Segundo, se debe ejecutar el comando "./waf install" para realizar una depuración en cada archivo ubicado en la ruta "src". Luego, se creará una carpeta de compilación llamada "ibex_opt" que contendrá el directorio de ejecución. En este directorio se ejecutará un archivo binario proporcionado por el framework desarrollado en *C++*. Es importante destacar que, para aplicar los cambios realizados en los scripts de Ibex, es necesario ejecutar dicho comando cada vez que se realicen modificaciones en el script.

```
root@50a963765986:/app# ./waf install
Waf: Entering directory `/app/__build__'
- install /usr/local/include/ibex/ibex_IntervalLibWrapper.h (from inter
val_lib_wrapper/gaol/ibex_IntervalLibWrapper.h)
- install /usr/local/share/pkgconfig/ibex.pc (from __build__/ibex.pc)
- install /usr/local/include/ibex/ibex_IntervalLibWrapper.inl (from int
erval_lib_wrapper/gaol/ibex_IntervalLibWrapper.inl)
- install /usr/local/include/ibex.h (from __build__/ibex.h)
- install /usr/local/include/ibex/ibex_LPLibWrapper.h (from lp_lib_wrap
per/soplex/ibex_LPLibWrapper.h)
- install /usr/local/include/ibex/ibex_Setting.h (from __build__/ibex_S
etting.h)
- install /usr/local/include/ibex/ibex_Dim.h (from src/arithmetic/ibex_
Dim.h)
- install /usr/local/include/ibex/ibex_Domain.h (from src/arithmetic/ib
```

Figura 17: Ejecución de comando install.

- Tercero, se ejecuta el script asociado al directorio build con ibex_opt. Además de su ejecución se almacena el dato en un archivo con extensión ".log" en donde entra el output de cada problema almacenado dentro del direcotrio "ibex_opt".

```
root@50a963765986:/app# ./__build__/src/ibexopt benchs/optim/medium/alk
ylbis.bch --random-seed=1 >> commands.log
```

Figura 18: Ejecución de comando ibex_opt.

```
********** setup **********
 file loaded:    benchs/optim/medium/alkylbis.bch
 random seed:    1
 output COV file:  benchs/optim/medium/alkylbis.cov
**************************************************

running.....

Tiempo de linealizacion: 0.0313411180001
Tiempo de ejecucion de la red: 0.627421906001
Tiempo de ejecucion de simplex: 0.215006122001
Precision a la -3, acotamiento del box: 0.000776610210841
Tiempo de ejecucion: 0.750585000001
Numero de nodos visitados: 86
Numero de variables de entrada: 14
Precicion a la -3 considerando las variables de entrada: 5.54721579173e-05
Preobabilidad aplicada sobre la red es de: 0.5
 results written in benchs/optim/medium/alkylbis.cov
 (old file saved in benchs/optim/medium/alkylbis.cov~)
```

Figura 19: Visualización de log file.

- Cuarto, la transferencia de información se realiza a través del productor asociado al software de mensajería en cola de Kafka. Esta elección se hizo con el objetivo de agilizar la transferencia de datos y aligerar la carga de procesamiento en cada equipo, a nivel de subprocesos del sistema operativo. Por lo tanto, se utilizarán hilos de ejecución (threads) para mantener un proceso de envío de datos en segundo plano, asegurando así la continua transmisión de información. Con ello además se utilizaron topics para segmentar los canales de envío de datos, aunque no está del todo terminado la segmentación por algunos problemas. A continuación una captura:

```
back > ibex-lib-master > producer.py > ...
1   from kafka import KafkaProducer
2   from json import dumps
3   import time
4   import threading
5   import argparse
6   import random
7   import string
8
9   servidores_bootstrap = 'kafka:9092'
10  topic_hard = 'hard'
11  topic_medium = 'medium'
12  topic_easy = 'easy'
13
14  productor = KafkaProducer(bootstrap_servers=[servidores_bootstrap])
15
16  def read_file(name_file):
17      data_from_ibex = []
18      name_file = name_file + '.log'
```

Figura 20: Kafka ibex producer.

- Kafka: vendría a ser el software de colas de mensajería que se utiliza para el enviado de información en “buffers” desde le producer ibex al consumer del “data process”. Cabe recalcar que se encuentran conectados mediante subredes establecidas por el mismo “yaml”; lo que también les permite vincular puertos locales con virtuales. Cabe recalcar que es de suma importancia el uso del *“Software”* zookeeper para el correcto funcionamiento de kafka.

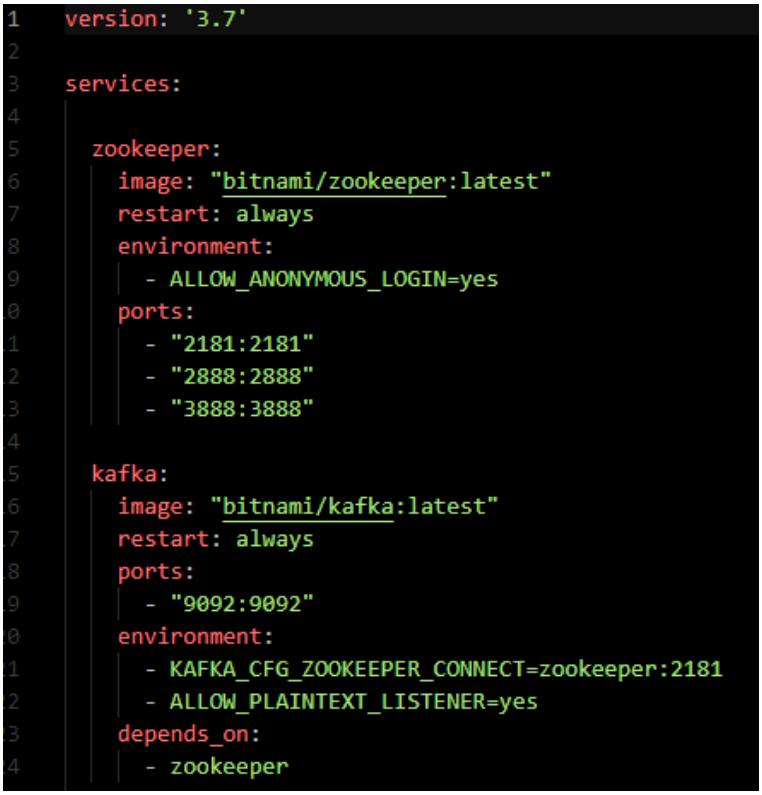

Figura 21: Kafka en archivo “docker-compose.yaml”.

- Topics: Son los canales en donde estará segmentada la información a partir de categorías como Hard, Easy y Medium; Cabe hacer mención que es sugerible aplicar algún tipo de patrón futuro para que al momento que entre un problema lo agrupo por algunas características, valide si efectivamente pertenece a alguna de estas categorías y así por último indique por qué canal le conviene redirigirse para su correcta transmisión.
- Consumer data processing: es el programa encargado de recibir la información y procesarla para luego insertarla en una base de datos liviana que utiliza el mismo protocolo que SQLite. Esta elección permite una validación más rápida sin necesidad de instalar otro servicio adicional, ya que SQLite se aloja en el mismo contenedor del consumidor. Posteriormente, la información puede ser consumida por el proceso de análisis de datos para su visualización en gráficas.
- Data_dump & Data_transfer: El programa que se encarga de leer la información procedente del consumer para realizar las gráficas y así mismo establecer algún tipo de comparativa a través de análisis por porcentaje, probabilidad, tiempo de ejecución, seed (a partir de este parámetro de entrada en ibex se puede identificar el tipo de distribución de probabilidad o tendencia la cual se estaría aplicando) y presición.

Fase experimental

A continuación se tomarán un problema por cada categoría y se evaluará mediante un número de 4 iteraciones para validar que tanto influye alguna probabilidad en específico con respecto al aprendizaje de la red neuronal. La probabilidad para este caso se dejaría constante netamente para observación de resultados.

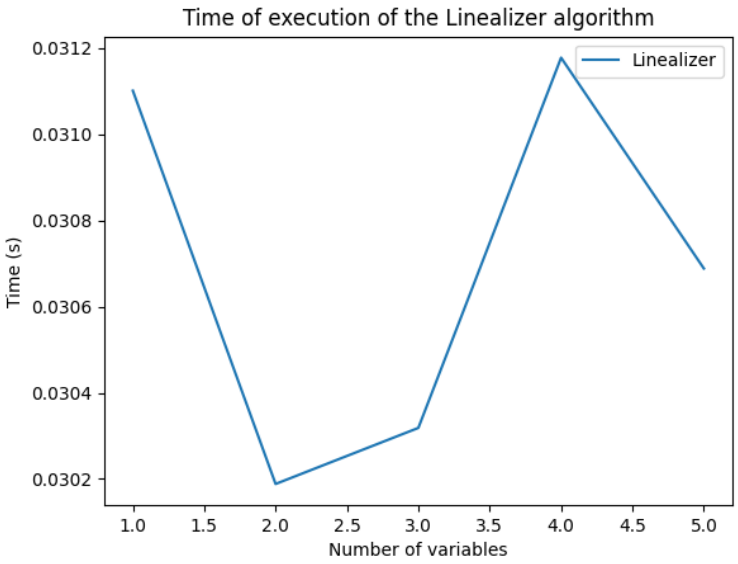

Figura 22: Linealización manteniendo la probabilidad constante.

Es evidente que existe un margen considerable en la diferencia entre cada tiempo. Observando los valores, que oscilan entre 0,0302 s y un *peack* de 0,0312 s, se puede deducir que los tiempos no son precisos, pero presentan una diferencia significativa en los intervalos. Por lo tanto, es posible que el porcentaje utilizado para el mejor caso no sea el más adecuado, lo que podría dar lugar a inconsistencias y una variabilidad notable. Con el fin de abordar esta situación, se propone almacenar la información en una base de datos, estructura o archivo para su posterior evaluación y análisis. De esta manera, se podrán seleccionar únicamente los mejores resultados basándose en el porcentaje y/o semilla que determinan la función de probabilidad.

A continuación se tiene una tabla con los valores entregados por cada iteración tanto de la solvencia del problema ya sea un “1” que sí es posible realizar una solución o un “0” señalando la conveniencia de dicha solución.

Para la primera iteración se obtuvieron los siguientes resultados, considerando una seed = 1:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0311006470001s
Tiempo de ejecucion de la red	0.542583573001s
Tiempo de ejecucion de simplex	0.209650670001s
Precision a la -3, acotamiento del box	0.000776610210841
Tiempo de ejecucion	0.726878000001s
Numero de nodos visitados	86
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	5.54721579173e-05
Probabilidad aplicada sobre la red es de	0.5

Para la segunda iteración se obtuvieron los siguientes resultados, considerando una seed = 1:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0301884930001s
Tiempo de ejecucion de la red	0.542583573001s
Tiempo de ejecucion de simplex	0.205950719001s
Precision a la -3, acotamiento del box	0.000776610210841
Tiempo de ejecucion	0.714070000001s
Numero de nodos visitados	86
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	5.54721579173e-05
Probabilidad aplicada sobre la red es de	0.5

Para la tercera iteración se obtuvieron los siguientes resultados, considerando una seed = 1:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0303185370001s
Tiempo de ejecucion de la red	0.549222940001s
Tiempo de ejecucion de simplex	0.205751972001s
Precision a la -3, acotamiento del box	0.000776610210841
Tiempo de ejecucion	0.735391000001s
Numero de nodos visitados	86
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	5.54721579173e-05
Probabilidad aplicada sobre la red es de	0.5

Para la cuarta iteración se obtuvieron los siguientes resultados, considerando una seed = 1:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0311772200001s
Tiempo de ejecucion de la red	0.606492575001s
Tiempo de ejecucion de simplex	0.207149818001s
Precision a la -3, acotamiento del box	0.000776610210841
Tiempo de ejecucion	0.731748000001s
Numero de nodos visitados	86
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	5.54721579173e-05
Probabilidad aplicada sobre la red es de	0.5

Para la quinta iteración se obtuvieron los siguientes resultados, considerando una seed = 1:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0306882890001s
Tiempo de ejecucion de la red	0.635477122001s
Tiempo de ejecucion de simplex	0.206969990001s
Precision a la -3, acotamiento del box	0.000776610210841
Tiempo de ejecucion	0.743855000001s
Numero de nodos visitados	86
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	5.54721579173e-05
Probabilidad aplicada sobre la red es de	0.5

Como puede apreciarse el valor no cambia por lo que ahora se optó por cambiar dicho valor sobre 3 problemas que se iterarán 5 veces cada uno para una comparación de métricas con además cambios en la probabilidad, precisión, tiempo

de ejecución y aprendizaje dado por la red neuronal MLP.

Los archivos con los que se trabajaran serán 3, uno por cada categoría y ellos vendrían a ser:

- easy/ackley_5.bch
- medium/alkylbis.bch
- hard/chembis.bch

Se iniciará con ackley_5.bch dentro de la categoría easy. En donde se obtuvieron los siguientes resultados:

Para la primera iteración se obtienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.00156540600001s
Tiempo de ejecucion de la red	0.0175088730001s
Tiempo de ejecucion de simplex	0.000752900000001s
Precision a la -3, acotamiento del box	2.57300368342e-08
Tiempo de ejecucion	0.0183080000001s
Numero de nodos visitados	22
Numero de variables de entrada	5
Precicion a la -3 considerando las variables de entrada	5.14600736684e-09
Probabilidad aplicada sobre la red es de	0.4

Para la segunda iteración se obtienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.00129612300001s
Tiempo de ejecucion de la red	0.0205858250001s
Tiempo de ejecucion de simplex	0.000418586000001s
Precision a la -3, acotamiento del box	2.57300368342e-08
Tiempo de ejecucion	0.0178170000001
Numero de nodos visitados	22
Numero de variables de entrada	5
Precicion a la -3 considerando las variables de entrada	5.14600736684e-09
Probabilidad aplicada sobre la red es de	0.210

Para la tercera iteración se obtienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.00170638300001s
Tiempo de ejecucion de la red	0.0320129600001s
Tiempo de ejecucion de simplex	0.000804849000001s
Precision a la -3, acotamiento del box	2.57300368342e-08
Tiempo de ejecucion	0.025502
Numero de nodos visitados	28
Numero de variables de entrada	5
Precicion a la -3 considerando las variables de entrada	5.14600736684e-09
Probabilidad aplicada sobre la red es de	0.5

Para la cuarta iteración se obtienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.00152754200001s
Tiempo de ejecucion de la red	0.0184028890001s
Tiempo de ejecucion de simplex	0.00114606800001s
Precision a la -3, acotamiento del box	2.57300368342e-08
Tiempo de ejecucion	0.0192560000001
Numero de nodos visitados	22
Numero de variables de entrada	5
Precicion a la -3 considerando las variables de entrada	5.14600736684e-09
Probabilidad aplicada sobre la red es de	0.01

Para la quinta iteración se obtienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.00169250300001s
Tiempo de ejecucion de la red	0.0198788090001s
Tiempo de ejecucion de simplex	0.000279203000001s
Precision a la -3, acotamiento del box	2.57300368342e-08
Tiempo de ejecucion	0.0178290000001
Numero de nodos visitados	24
Numero de variables de entrada	5
Precicion a la -3 considerando las variables de entrada	5.14600736684e-09
Probabilidad aplicada sobre la red es de	0.4325

Se puede observar una tendencia consistente en los resultados, lo cual indica que la red está experimentando overfitting. Esto significa que siempre obtiene los mejores resultados y los valores se mantienen constantes en varios casos, incluso cuando se modifican los porcentajes, la precisión, etc. Este patrón puede ser un indicador y, al mismo tiempo, un problema al identificar problemas de categoría fácil. Para garantizar resultados más confiables, sería conveniente probar con otros modelos disponibles en la biblioteca Keras.

A continuación se tienen las siguientes capturas procedentes de las gráficas, en donde se dará atención especialmente al de precisión que se mantiene constante como una pendiente al eje x.

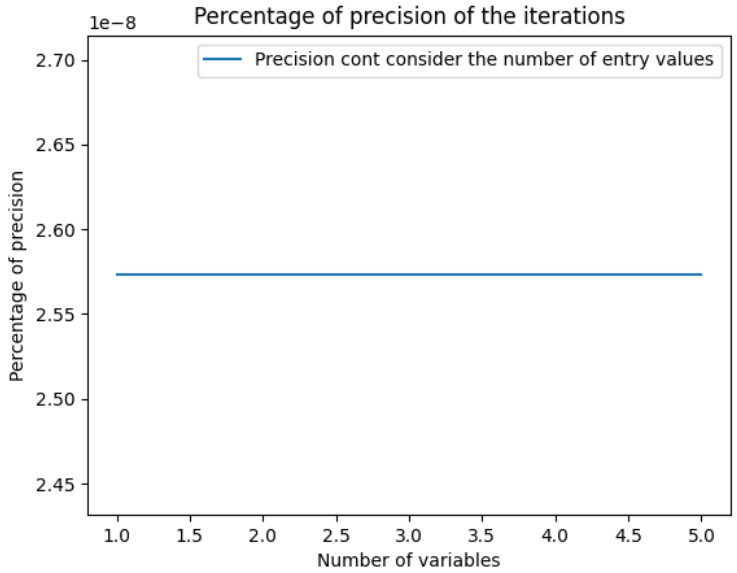

Figura 23: Precisión para problemas easy.

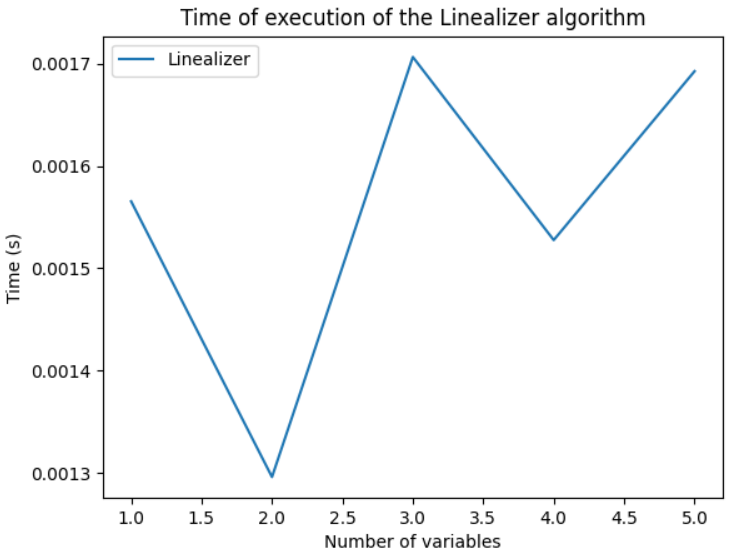

Figura 24: Tiempo de linealización easy.

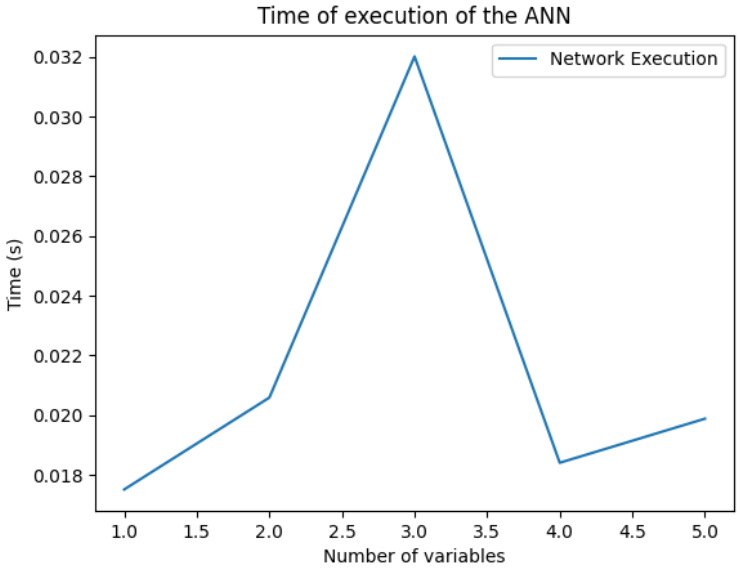

Figura 25: Tiempo de MLP para problemas easy.

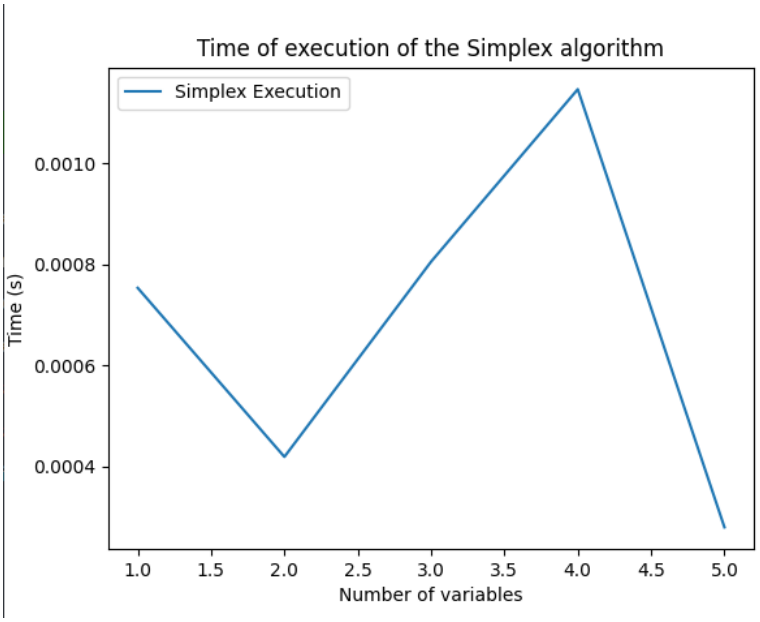

Figura 26: Tiempo de ejecución Simplex para problemas easy.

Para el caso de la categoría medium se dará uso a “alkylbis.bch”. En donde se obtuvieron los siguientes resultados:

Para la primera iteración se tienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0206720790001s
Tiempo de ejecucion de la red	0.558092955001s
Tiempo de ejecucion de simplex	0.150827633001s
Precision a la -3, acotamiento del box	2.95320958552e-05
Tiempo de ejecucion	0.518288000001s
Numero de nodos visitados	62
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	2.10943541823e-06
Probabilidad aplicada sobre la red es de	0.02

Para la segunda iteración se tienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0218346080001s
Tiempo de ejecucion de la red	0.512725123001s
Tiempo de ejecucion de simplex	0.146546427001s
Precision a la -3, acotamiento del box	0.00010619843323
Tiempo de ejecucion	0.529080000001s
Numero de nodos visitados	60
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	7.58560237352e-06
Probabilidad aplicada sobre la red es de	0.210

Para la tercera iteración se tienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0165140420001s
Tiempo de ejecucion de la red	0.364417935001s
Tiempo de ejecucion de simplex	0.121254475001s
Precision a la -3, acotamiento del box	0.5
Tiempo de ejecucion	0.415698000001s
Numero de nodos visitados	44
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	7.14285714286e-05
Probabilidad aplicada sobre la red es de	0.32

Para la cuarta iteración se tienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0221773420001s
Tiempo de ejecucion de la red	0.513259493001s
Tiempo de ejecucion de simplex	0.149986162001s
Precision a la -3, acotamiento del box	0.00010619843323
Tiempo de ejecucion	0.552247000001s
Numero de nodos visitados	60
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	7.58560237352e-06
Probabilidad aplicada sobre la red es de	0.4135

Para la quinta iteración se tienen los siguientes resultados:

Parámetros de medición	**Resultado**
Tiempo de linealización	0.0200189850001s
Tiempo de ejecucion de la red	0.452759055001s
Tiempo de ejecucion de simplex	0.144760759001s
Precision a la -3, acotamiento del box	2.95320958552e-05
Tiempo de ejecucion	0.531866000001s
Numero de nodos visitados	62
Numero de variables de entrada	14
Precicion a la -3 considerando las variables de entrada	2.10943541823e-06
Probabilidad aplicada sobre la red es de	0.1999

A continuación se realizarán una comparativa de gráficas considerando tiempo de linealización, tiempo de ejecución de la red MLP, tiempo de ejecución del Simplex, la precisión considerando un margen de 10 elevado a -3, y la precisión considerando la cantidad de variables de entrada.

Cabe recalcar que el porcentaje favorable para este caso vendría a ser 0,210 en este ejercicio ya que ofrece una presición por sobre 50 %, mientras que las inferiores o superiores la precisión tiende a perderse. Lo cúal también cumple para el caso de consideración de los valores de entrada ya que el porcentaje al ser menor, de igual forma aumenta independiente que la subida sea ínfima.

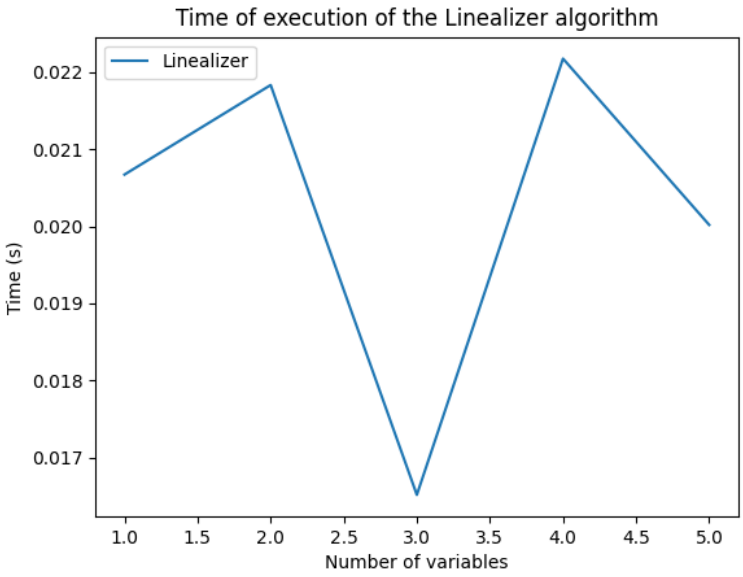

Figura 27: Linealización para problema medium.

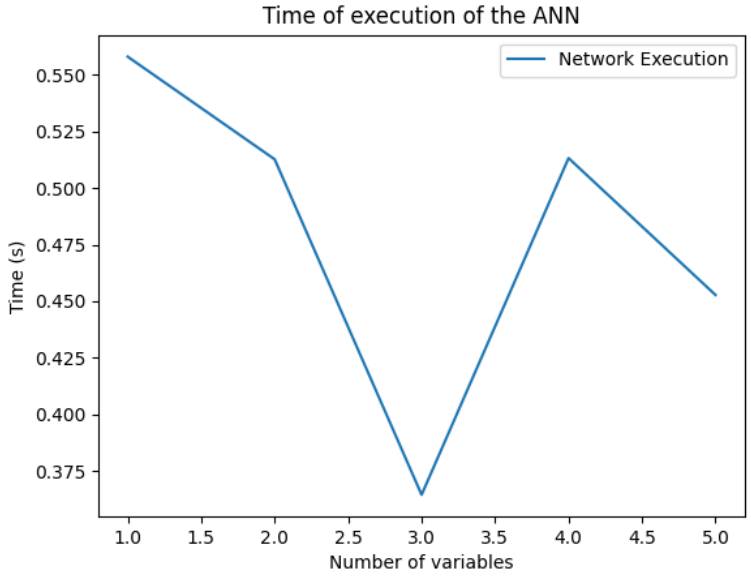

Figura 28: Ejecución de red MLP para problema medium.

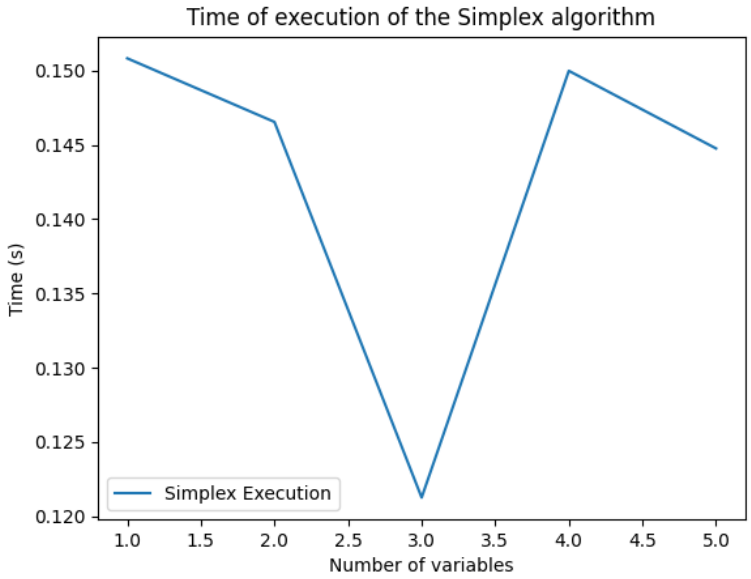

Figura 29: Ejecución de Simplex para problema medium.

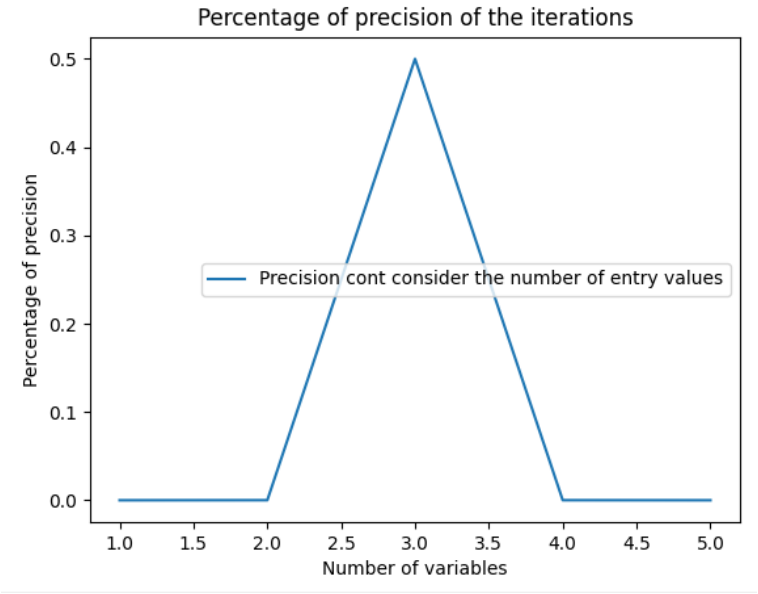

Figura 30: Precisión para problema medium.

Es importante destacar que la rapidez de ejecución del algoritmo simplex no garantiza necesariamente su precisión. Para lograr una validación adecuada, es fundamental identificar el porcentaje apropiado, teniendo en cuenta el modelo y el problema específico. Esto tiene un impacto directo en la cantidad de vértices acotados por la solución, como se puede apreciar en las tablas. En este caso particular, se ha determinado que 60 nodos son ideales para obtener una precisión óptima. Sin embargo, es relevante tener en cuenta que, al evaluarlo con otros modelos, este número podría variar, ya que dependerá del análisis detallado de cada resultado.

Para los problemas de categoría “hard”, se ha observado que el sistema enfrenta dificultades en cuanto a la compilación de todos los problemas, lo cual resulta en un tiempo considerablemente mayor para la redacción de las soluciones. Se sospecha que este problema podría estar relacionado con el modelo utilizado, pero también se considera que la limitación de la red para manejar problemas de gran complejidad puede estar influyendo en este contratiempo. Actualmente, se está llevando a cabo un seguimiento detallado para identificar las causas exactas y encontrar una solución adecuada antes de la presentación. Se espera que para ese momento, se cuente con una respuesta eficiente y satisfactoria para este inconveniente.

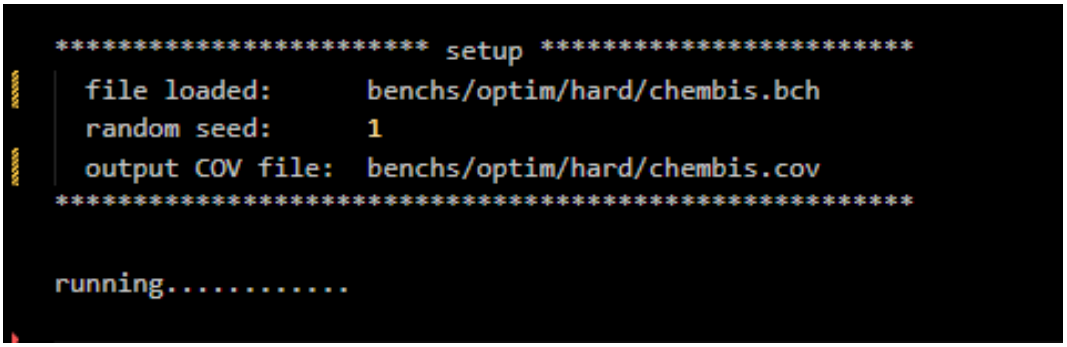

Figura 31: Problema hard con inconvenientes de ejecución.

6. Conclusión

Al considerar la precisión y las variables en el contexto del overfitting, se puede apreciar la importancia de encontrar un equilibrio óptimo. Si se obtienen resultados consistentemente altos pero sin variación significativa al modificar los parámetros, es probable que la red esté experimentando overfitting, lo cual puede ser problemático al abordar problemas de categoría fácil. Para abordar este desafío, es recomendable explorar otros modelos disponibles en la biblioteca Keras, lo que permitirá obtener resultados más confiables y adaptados al problema específico.

Además, la integración de colas de mensajería en el sistema presenta beneficios notables. Al utilizar un sistema de colas de mensajería, se logra una mayor agilidad en el intercambio de recursos y se evitan interrupciones en el procesamiento al compartir información entre diferentes componentes. Esto permite una mejor utilización de los recursos disponibles y facilita el procesamiento de problemas en algoritmos simples con redes neuronales MLP (Multi-Layer Perceptron).

Para los problemas de categoría "hard" se pudo apreciar inconvenientes al momento de procesamiento esto puede ser causado por el modelo y la carga que recibe la red neuronal; el cálculo puede intuirse que puede llegar a demorarse quizás 45 minutos consideran la alta complejidad del problema a evaluar.

En resumen, al considerar la precisión, las variables y el overfitting, es fundamental encontrar un equilibrio adecuado para obtener resultados confiables. La exploración de diferentes modelos en la biblioteca Keras permitirá abordar de manera más efectiva los desafíos relacionados con el overfitting. Asimismo, la integración de colas de mensajería en el sistema proporciona una solución ágil y eficiente para compartir recursos y evitar interrupciones en el procesamiento. Esto mejora la capacidad de procesamiento de problemas en algoritmos simples con redes neuronales MLP.

7. Bibliografía

- Paper1: Anteproyecto: Entrega de propuesta por Memorista Nicolás Calderón a cargo de Victor Reyes
- Paper2: Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects, por Ignacio Araya y Victor Reyes.
  `https://www.researchgate.net/publication/288918049_Interval_Branch-and-Bound_algorithms_for_optimization_and_constraint_satisfaction_a_survey_and_prospects`
- Paper3: THE SIMPLEX-SIMULATED ANNEALING APPROACH TO CONTINUOUS NON-LINEAR OPTIMIZATION por MARGARIDA F. CARDOSO, R. L. SALCEDOt and S. FEvo DE AZEVEOO
  `https://paginas.fe.up.pt/~sfeyo/Docs_SFA_Publica_Conferences/SFA_JP_19960101_CCE_The%20Simplex-Simulated%20Annealing.pdf`
- Paper4: Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks por Sophia Voulgaropoulou, Nikolaos Samaras y Nikolaos Ploskas
  `https://www.researchgate.net/publication/359465464_Predicting_the_Execution_Time_of_the_Primal_and_Dual_Simplex_Algorithms_Using_Artificial_Neural_Networks`

- Paper5: Nonlinear Models and Problems in Applied Sciences from Differential Quadrature to Generalized Collocation Methods por N. BELLOMO.
  https://www.sciencedirect.com/science/article/pii/S0895717797001428