UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación y Facultad de Ciencias

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Combinando distintas técnicas para el diseño de una metaheurística para problemas de optimización de alta dimensionalidad

Presentado por:
Jesús García León

Responsable de tutorización
Daniel Molina Cabrera
Departamento de Ciencias de la Computación e Inteligencia Artificial

Curso académico 2024-2025

Declaración de originalidad

D./Dña. Jesús García León

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 24 de noviembre de 2024

Fdo: Jesús García León

Agradecimientos

Quiero agradecer en primer lugar a mis padres, ya que sin su esfuerzo y apoyo nada de este trabajo habría sido posible. A mis compañeros de clase por permanecer a mi lado durante todo el camino, motivarme a aprender más cada día para no quedarme atrás, por enseñarme y por dejarse enseñar. A mis amigos por todo el apoyo incondicional que me han dado en los buenos y malos momentos, un apoyo no solo académico, sino moral, que ha sido necesario para poder llegar hasta aquí. Por último quiero agradecer a mi tutor, por aconsejarme y guiarme a lo largo de este trabajo.

Resumen

El aumento de la capacidad de cálculo de los ordenadores ha posibilitado resolver problemas cada vez más complejos, que dependen de un número creciente de variables. La utilización de algoritmos clásicos en este contexto no es posible o resulta extremadamente costosa debido al incremento exponencial en la dificultad que trae consigo la alta dimensionalidad. Esto hace conveniente diseñar algoritmos capaces de encontrar soluciones para problemas de alta dimensionalidad. Este problema se conoce como *la maldición de la alta dimensionalidad*. Este fenómeno es el aumento exponecial del espacio de búsqueda que se produce al aumentar la dimensionalidad de un problema, lo que dificulta gravemente la búsqueda de soluciones.

En problemas de alta dimensionalidad, muchas técnicas que funcionan eficientemente en espacios de baja dimensión se vuelven ineficaces, ya que los datos ocupan solo una pequeña fracción del espacio total. Esto afecta algoritmos de búsqueda, clasificación y optimización, ya que el aumento del espacio de búsqueda, trae consigo un aumento del tiempo necesario, que hace inviable evaluar todas las posibles opciones.

En este trabajo proponemos el estudio e implementación de varias técnicas de **agrupamiento de variables**, que busca separar las variables de un problema en subgrupos independientes, permitiendo optimizar cada subgrupo por separado. Esto reduce significativamente la dimensionalidad efectiva del problema, permitiendo una optimización más eficiente. Al dividir el conjunto de variables en subgrupos independientes, aprovechamos el paralelismo y reducimos la complejidad computacional, ya que cada subgrupo puede optimizarse sin afectar a los demás. Esta técnica es especialmente útil en problemas donde ciertas variables están altamente correlacionadas entre sí, pero tienen poca o ninguna relación con otras variables del conjunto.

Estudiaremos esta técnica aplicándola a algoritmos diseñados para resolver problemas de alta dimensionalidad y a algoritmos que funcionan mejor en baja dimensión. Así, podremos comparar si es más eficiente diseñar algoritmos que mejoren en alta dimensionalidad o dividir el problema en subproblemas menores. Para ello, introduciremos los conceptos matemáticos necesarios para entender cómo resolver problemas de minimización, así como la teoría detrás de los algoritmos de agrupamiento de variables. Además, analizaremos qué tests estadísticas podemos emplear para obtener una comparación objetiva y efectiva, explicando su selección.

Presentaremos también algoritmos metaheurísticos básicos que más adelante combinaremos para crear nuestra propuesta. Entre ellos, se incluyen *SHADE* y *SHADE-ILS* como algoritmos de optimización, y *ERDG* como algoritmo de agrupamiento de variables.

Adicionalmente, desarrollaremos una biblioteca en el lenguaje de programación *Julia* que implemente los algoritmos utilizados. Esta biblioteca no solo tiene como objetivo evaluar la efectividad de la técnica de agrupamiento de variables, sino también contribuir a la comunidad de *Julia*, proporcionando una herramienta que otros desarrolladores podrán utilizar y mejorar en el futuro. Para comparar los resultados, utilizaremos el benchmark *CEC2013 LSGO*, diseñado específicamente para problemas de optimización global de alta dimensionalidad.

Palabras clave: capacidad de cálculo, optimización, redes neuronales, alta dimensionalidad,

maldición de la dimensionalidad, agrupamiento de variables, complejidad computacional, metaheurísticas, Julia.

Abstract

The increase in computational power of computers has enabled the resolution of increasingly complex problems that depend on a growing number of variables. The use of classical algorithms in this context is either infeasible or extremely costly due to the exponential growth in difficulty brought by high dimensionality. This makes it necessary to design algorithms capable of finding solutions to high-dimensional problems. This issue is known as the *curse of high dimensionality*. This phenomenon refers to the exponential growth of the search space as the dimensionality of a problem increases, which severely hinders the ability to find solutions.

In high-dimensional problems, many techniques that work efficiently in low-dimensional spaces become ineffective, as data occupies only a small fraction of the total space. This impacts search, classification, and optimization algorithms, as the growth in the search space leads to an increase in the required computation time, making it impractical to evaluate all possible options.

In this work, we propose the study and implementation of various **variable grouping** techniques, which aim to separate the variables of a problem into independent subgroups, allowing each subgroup to be optimized separately. This significantly reduces the effective dimensionality of the problem, enabling more efficient optimization. By dividing the set of variables into independent subgroups, we leverage parallelism and reduce computational complexity, as each subgroup can be optimized without affecting the others. This technique is particularly useful in problems where certain variables are highly correlated with each other but have little or no relationship with other variables in the set.

We will study this technique by applying it to algorithms designed to solve high-dimensional problems as well as algorithms that perform better in low dimensions. This will allow us to compare whether it is more efficient to design algorithms that improve performance in high dimensions or to divide the problem into smaller subproblems. To this end, we will introduce the mathematical concepts necessary to understand how to solve minimization problems, as well as the theory behind variable grouping algorithms. Additionally, we will analyze which statistical tests can be used to achieve an objective and effective comparison, explaining the rationale for their selection.

We will also present basic metaheuristic algorithms that we will later combine to develop our proposal. Among these are *SHADE* and *SHADE-ILS* as optimization algorithms, and *ERDG* as a variable grouping algorithm.

Moreover, we will develop a library in the *Julia* programming language to implement the algorithms used. This library aims not only to evaluate the effectiveness of the variable grouping technique but also to contribute to the *Julia* community by providing a tool that other developers can use and improve in the future. To compare results, we will use the *CEC2013 LSGO* benchmark, specifically designed for global optimization problems in high dimensions.

Keywords: computational power, optimization, neural networks, high dimensionality, curse of dimensionality, variable grouping, computational complexity, metaheuristics, Julia.

Índice general

Agradecimientos III

Resumen V

Abstract VII

Introducción XI

Repaso Bibliográfico XIII
- 0.1. Enfoques Principales en LSGO . . . XIII
- 0.2. Explotación de la Estructura del Problema (evitando la caja negra) . . . XIII
- 0.3. Hibridación y Algoritmos Meméticos . . . XV
- 0.4. Operadores de Muestreo y Variación en DE y PSO . . . XVI

Presupuesto XIX

I. Parte Matemática 1

1. Preliminares 3
- 1.1. Conceptos matemáticos fundamentales . . . 3

2. Optimización Numérica 5
- 2.1. Optimización Sin Restricciones . . . 5
- 2.2. Optimización Con Restricciones . . . 6
- 2.3. Condiciones para Óptimos Locales . . . 6
  - 2.3.1. Condiciones de Primer Orden . . . 6
  - 2.3.2. Condiciones de Segundo Orden . . . 7
- 2.4. Métodos de Búsqueda en Optimización . . . 7
  - 2.4.1. Métodos de Búsqueda en Línea . . . 7
- 2.5. Métodos de Región de Confianza . . . 10
- 2.6. Métodos Quasi-Newton . . . 10
  - 2.6.1. Actualización del Hessiano . . . 10
  - 2.6.2. El Algoritmo BFGS . . . 11
- 2.7. Optimización en alta dimensionalidad . . . 12
  - 2.7.1. El Algoritmo L-BFGS . . . 12
  - 2.7.2. Ventajas del L-BFGS . . . 13

3. Agrupamiento de variables 15
- 3.1. Definiciones . . . 15
- 3.2. Teoremas . . . 16
  - 3.2.1. Agrupamiento diferencial . . . 17
  - 3.2.2. Agrupamiento diferencial recursivo . . . 18

4. Tests estadísticos **21**
4.1. Comparaciones por pares . . . 21
4.1.1. Prueba de signos . . . 22
4.1.2. Prueba de rangos con signo de Wilcoxon . . . 22
4.2. Comparaciones múltiples con un método de control . . . 23
4.2.1. Prueba de Friedman y extensiones . . . 24
4.2.2. Procedimientos post-hoc . . . 26

II. Parte Informática **29**

5. Componentes de los algoritmos implementados **31**
5.1. Evolución diferencial . . . 31
5.2. Búsqueda local . . . 32
5.3. Algoritmos de descomposición . . . 32

6. Algoritmos de comparación **33**
6.1. SHADE . . . 33
6.2. SHADE-ILS . . . 37
6.3. DG2 . . . 38
6.4. ERDG . . . 38

7. Propuesta **39**
7.1. ERDG-SHADE . . . 39
7.2. ERDG-SHADE-ILS . . . 39

8. Resultados **41**
8.1. Resultados obtenidos . . . 41

9. Conclusiones **43**
9.1. Conclusiones extraídas del análisis de los datos . . . 43

Bibliografía **45**

Introducción

Cuando enfrentamos un problema de optimización que depende de pocos factores, es sencillo idear estrategias para resolverlo y encontrar una solución. Sin embargo, a medida que crece el número de variables, la complejidad del problema aumenta, y se hace inviable resolverlo mediante métodos tradicionales. Para abordar esta problemática, se han diseñado algoritmos específicos para manejar la alta dimensionalidad utilizando metaheurísticas. Estas buscan optimizar la exploración del espacio de búsqueda, logrando un equilibrio entre exploración y explotación, lo que permite realizar búsquedas efectivas en un espacio donde la porción representada por la población de búsqueda es ínfima comparada con el espacio total. No obstante, estos algoritmos presentan limitaciones, y cuando el número de variables es suficientemente alto, hay dos opciones, por un lado, las metaheurísticas especialmente diseñado para LSGO, como SHADE-ILS, y por el otro, técnicas que permitan reducir la dimensionalidad que aborda el algoritmo.

En este trabajo, proponemos utilizar técnicas de **agrupamiento diferencial automático de variables**, que permite identificar las variables que interactúan entre sí, para asignarles el mismo grupo. De esta forma somos capaces de reducir el problema en subproblemas, uno para cada subgrupo ya que cada variable solo interactúa con las de su propio grupo y no (o mínimamente) con las de otros grupos. Así, podemos reducir la dimensión efectiva del problema, convirtiendo un problema de optimización de gran escala en problemas de optimización de menor tamaño. Nuestro objetivo es hibridar esta técnica con algoritmos diseñados para la alta dimensionalidad y con algoritmos que no están específicamente adaptados para ello. Compararemos el rendimiento aplicando y no aplicando esta técnica en ambos tipos de algoritmos.

Para probar la efectividad de esta técnica, se han planteado los siguientes objetivos:

Objetivos

Parte matemática

- **Análisis teórico**: Repaso de las técnicas matemáticas tradicionales en la literatura para resolver problemas de optimización sin restricciones.
- **Estudio de teoremas**: Exposición y análisis de los teoremas en los que se basa el agrupamiento diferencial.
- **Fundamentos estadísticos**: Explicación de los tests estadísticos utilizados para la comparación de algoritmos.

Parte informática

- **Desarrollo de una biblioteca de técnicas de agrupamiento**: Implementación de algoritmos para optimización continua sin restricciones en el lenguaje de programación Julia y de técnicas de agrupamiento automático de variables.

- **Selección de algoritmos**: Descripción de los algoritmos que se incluirán en la biblioteca y que se emplearán en las comparaciones.
- **Evaluación experimental**: Análisis de los algoritmos en el benchmark *cec2013lsg0*, presentación de resultados y conclusiones obtenidas.

Estructura de la memoria

El trabajo se divide en tres partes claramente diferenciadas y cada parte está dividida en capítulos. Ademas de las dos partes que constituyen el contenido de este TFG, tenemos esta primera sección introductoria, donde se pone en contexto el trabajo, se definen los objetivos y se hace un repaso de la bibliografía necesaria. Además, se hace una estimación del presupuesto y del tiempo dedicado a cada tarea.

La parte Matemática está dividida en 3 capítulos, un primer capítulo donde se realizan definiciones importantes sobre optimización de funciones, se exponen algunos teoremas necesarios y se explican los métodos de búsqueda lineal y el algoritmo L-BFGS-B, que será un componente importante de nuestro algoritmo final. El segundo capítulo de esta sección se define el agrupamiento diferencial y se citan y demuestran los teoremas más importantes que justifican el diseño y utilidad de los algoritmos de agrupamiento automático de variables. Además se proporcionan algunas definiciones previas necesarias sobre separabilidad de funciones. Por último en la tercera sección de esta parte, se explican los fundamentos de los tests estadísticos y se definen los tests que suelen ser utilizados para comparar algoritmos.

En la parte informática, tenemos un capitulo inicial donde se definen los componentes de los algoritmos que formarán nuestra propuesta, una segunda parte donde se explican los algoritmos que luego hibridaremos con el agrupamiento de variables. En tercer lugar se explicará nuestra propuesta: combinar ERDG con SHADE y SHADE-ILS. Por último, unas secciones finales donde se exponen los resultados obtenidos y las conclusiones que hemos obtenido de los resultados.

Repaso Bibliográfico

En esta sección, haremos un repaso bibliográfico sobre la optimización global a gran escala (LSGO, por sus siglas en inglés, *Large Scale Global Optimization*). Se proporcionará un contexto general de las distintas técnicas metaheurísticas y enfoques que existen para abordar este tipo de problemas, y se situarán nuestros algoritmos en este contexto, explicando en mayor profundidad aquellas técnicas que están más relacionadas con los algoritmos *SHADE* [12], *SHADE-ILS* [5], *ERDG* [15] y *DG2* [7]. Para ello nos basaremos en el estudio realizado por Omidvar et al. en [8] y [9].

0.1. Enfoques Principales en LSGO

A continuación, se presentan los enfoques principales en LSGO, los cuales se han desarrollado para afrontar la complejidad de los problemas de optimización de gran escala:

1. **Descomposición del problema**: Dividir el problema en subproblemas manejables.
2. **Hibridación y búsqueda local memética**: Combina algoritmos evolutivos con técnicas de búsqueda local.
3. **Operadores de muestreo y variación**: Técnicas de muestreo para explorar el espacio de búsqueda.
4. **Modelado por aproximación y uso de modelos sustitutos**: Se utilizan modelos simplificados para reducir el coste computacional.
5. **Métodos de inicialización**: Métodos para asegurar una cobertura uniforme del espacio de búsqueda.
6. **Paralelización**: Uso de múltiples instancias de algoritmos para acelerar la búsqueda.

0.2. Explotación de la Estructura del Problema (evitando la caja negra)

Explotar la estructura del problema es común en diversas áreas de la optimización. La optimización de tipo *gray-box* es un concepto relativamente nuevo que se centra en incorporar la estructura del problema en el proceso de optimización. Este enfoque busca descubrir y explorar un "orden oculto", un concepto ampliamente estudiado en computación evolutiva en el contexto del aprendizaje de dependencias (*linkage learning*). Las dependencias entre variables son cruciales para el rendimiento de los algoritmos genéticos (GAs). Sin conocer estas dependencias, incluso problemas separables pueden ser exponencialmente difíciles para GAs simples, y su resolución puede requerir tamaños de población exponencialmente grandes para encontrar el óptimo global.

En la optimización *gray-box*, se asume que la estructura del problema es conocida *a priori*, como ocurre en problemas discretos y combinatorios.

Sin embargo, en problemas continuos, la estructura puede no ser evidente, por lo que primero debe ser descubierta. Se han propuesto diversos algoritmos para analizar la interacción

entre variables y capturar la topología del problema, lo cual no solo convierte problemas de tipo *black-box* en *gray-box*, sino que también permite revelar información no trivial, incluso en problemas completamente conocidos (*white-box*).

La información estructural puede emplearse de diferentes formas para mejorar el rendimiento en la optimización. Algunos métodos, como la *cooperative co-evolution* (CC) [4], descomponen el problema en subproblemas de menor dimensionalidad, requiriendo un análisis de interacción de variables para lograr una descomposición adecuada. Otros métodos, como los algoritmos evolutivos basados en distribuciones (*EDAs*) y los algoritmos de optimización bayesiana (*BOAs*), no descomponen el problema explícitamente, pero capturan y aprovechan la información de interacción mediante modelos probabilísticos construidos durante el proceso de optimización.

Para descubrir la estructura interna del problema, existen dos aproximaciones, los métodos implícitos y los métodos explícitos. A continuación se presenta una clasificación de ambos métodos:

Método	**Descripción**
Adaptación de Interacciones	Métodos que extienden algoritmos genéticos (GAs) simples agregando mecanismos para mejorar la representación del problema y promover una vinculación estrecha (*tight linkage*).
Modelos Probabilísticos	Métodos que utilizan distribuciones de probabilidad para representar la función objetivo o características del problema.
Submétodos:	
- *Construcción de un modelo de interacción de variables*	Los *EDAs* y *BOAs* modelan las interacciones entre variables mediante redes bayesianas o matrices de covarianza adaptativas. Este enfoque captura patrones complejos en el problema.
- *Construcción de un modelo de la función objetivo*	La optimización bayesiana construye un modelo probabilístico de la función objetivo basado en distribuciones previas y posteriores, que se actualizan con nuevas evaluaciones.
- *Construcción de un modelo del movimiento poblacional*	Estrategias como *CMA-ES* modelan el movimiento poblacional utilizando distribuciones gaussianas multivariantes, ajustando la matriz de covarianza para reflejar el paisaje del problema.
Reducción Dimensional y Partición del Espacio	Técnicas que reducen la dimensionalidad o dividen el espacio en subespacios más pequeños para simplificar la complejidad computacional en problemas de gran escala. Ejemplos incluyen proyecciones aleatorias y análisis de componentes principales (PCA).
Distribuciones de Cola Pesada	Uso de distribuciones como Lévy, Cauchy o t-distributions para mejorar la exploración y la diversidad poblacional. Estas distribuciones son útiles para evitar el estancamiento en óptimos locales.

Tabla 1.: Resumen de métodos implícitos para la explotación de la estructura del problema en optimización.

Método	Descripción
Descomposición en Coevolución Cooperativa (CC)	Divide el problema en subproblemas de menor dimensionalidad, optimizando cada uno por separado en un esquema de coevolución.
Submétodos:	
- *CCGA (Cooperative Coevolution Genetic Algorithm)*	Divide un problema de n dimensiones en n subproblemas unidimensionales. Útil para problemas de baja interacción entre variables.
- *Divide-in-half*	Divide el problema en dos subcomponentes de igual tamaño, optimizados iterativamente con estrategias como DE (*Differential Evolution*).
Gestión de Interacciones entre Variables	Métodos que identifican relaciones entre variables para minimizar interacciones entre componentes.
Submétodos:	
- *Agrupamiento Aleatorio (Random Grouping)*	Reorganiza aleatoriamente variables después de cada ciclo evolutivo para aumentar la probabilidad de agrupar variables relacionadas.
- *Agrupamiento Delta (Delta Grouping)*	Ordena variables según sus desplazamientos medios entre iteraciones, agrupándolas por magnitud similar.
- *Minimización de Diferencias de Fitness (DI)*	Reorganiza variables para minimizar diferencias en las interacciones detectadas, optimizando componentes uniformes.
Métodos Estadísticos	Inferencia de interacciones utilizando análisis estadístico de la población en evolución.
Métodos Basados en Diferencias Finitas	Utilizan diferencias finitas para detectar interacciones entre pares de variables.
Submétodos:	
- *DG (Differential Grouping)*	Forma componentes no separables mediante un análisis iterativo de interacciones con menor coste computacional.
Agrupamiento Automático y Semiautomático	Algoritmos que forman grupos de forma automática o requieren que el usuario especifique el número/tamaño de los componentes.
Submétodos:	
- *Automático*	Métodos como *DG2* usan matrices de interacción y algoritmos de componentes conectados para agrupar.
- *Semiautomático*	Requieren información adicional, como el número o tamaño de componentes, para formar grupos.
- *k-s Dimensional Components*	Métodos que requieren que el usuario especifique tanto el número como el tamaño de los componentes para formar los grupos.

Tabla 2.: Resumen de métodos explícitos para la explotación de la estructura del problema en optimización.

Tras haber establecido una clasificación, podemos observar que DG2 y ERDG pertenecen a la categoría de algoritmos explícitos, basados en diferencias finitas y automáticos.

0.3. Hibridación y Algoritmos Meméticos

El teorema de No Free Lunch [13] indica que ningún algoritmo de búsqueda puede superar consistentemente a todos los demás en todos los problemas posibles. La **hibridación** busca combinar las fortalezas de diferentes algoritmos para mejorar su rendimiento, la calidad de las soluciones y su integración en sistemas más amplios.

- Algoritmos de Búsqueda Local Híbrida Estos algoritmos se basan únicamente en búsqueda local sin un componente global explícito. Por ejemplo, el *Multiple Trajectory Search* (MTS) emplea tres métodos de búsqueda local, seleccionando el mejor para optimizar en vecindades específicas. Este enfoque ha demostrado eficacia en problemas de hasta 1000 dimensiones.

- Algoritmos Meméticos Integran búsqueda local dentro de un marco evolutivo global, balanceando exploración y explotación. Son populares en la optimización de gran escala, destacándose en competiciones de LSGO. Un ejemplo es el algoritmo SHADE-ILS, que utilizaremos en nuestra propuesta. Los principales aspectos de diseño incluyen:

  - **Frecuencia de búsqueda local:** Puede ser fija, adaptativa o probabilística.
  - **Selección de soluciones:** Basada en rendimiento, aleatoriedad o todas las soluciones.
  - **Intensidad de búsqueda:** Determina la duración de la búsqueda local, con enfoques fijos o adaptativos.
  - **Procedimientos de búsqueda local:** Amplia variedad, incluyendo métodos como MTS-LS, L-BFGS-B.

- Hibridación Coevolutiva Combina la descomposición de problemas con algoritmos meméticos. Los problemas se dividen en subproblemas optimizados con algoritmos globales seguidos de episodios de búsqueda local.

0.4. Operadores de Muestreo y Variación en DE y PSO

Los operadores de muestreo y variación buscan mantener la diversidad en la población y mejorar la eficacia de los algoritmos en la exploración de grandes espacios de búsqueda. Dos enfoques comunes son:

- Evolución Diferencial (DE)

  La Evolución Diferencial (DE) [10] es un algoritmo popular en la optimización global debido a su simplicidad y efectividad. Variantes como *SHADE* y *SHADE-ILS* han surgido como adaptaciones de DE para problemas de gran escala:

  - **SHADE**: Una variante de DE que ajusta adaptativamente el tamaño de la población y los parámetros de mutación para mantener la diversidad en poblaciones grandes.
  - **SHADE-ILS**: Extiende SHADE mediante la integración de estrategias de búsqueda local, mejorando la precisión en problemas de alta dimensionalidad.

- Particle Swarm Optimization (PSO)

  PSO [2] es un método basado en el comportamiento social de partículas. Aunque efectivo en problemas de baja dimensionalidad, enfrenta retos en alta dimensionalidad, para lo cual se han introducido estrategias de *mantenimiento de diversidad* y *re-inicialización*.

Método	Descripción
Estrategias de mutación en DE	Variaciones de la estrategia de mutación para mejorar la convergencia y diversidad en problemas de gran escala.
Submétodos:	
- *Adaptación de estrategias de mutación*	Aplicación adaptativa de estrategias según el tipo de problema, como DE/rand/1 o DE/current-to-best/1.
- *Selección de vectores*	Uso de vectores basados en calidad o cercanía, como la combinación de global-best y personal-best.
Adaptación de parámetros en DE	Ajuste dinámico de parámetros como el factor de escala (F) y la tasa de cruce (CR) para mejorar la exploración y explotación.
Submétodos:	
- *Muestreo probabilístico*	Uso de distribuciones (uniforme, gaussiana, Cauchy) para generar parámetros adaptativos.
- *Procesos caóticos*	Ajuste de F y CR mediante procesos caóticos para mejorar la búsqueda en espacios complejos.
Mantenimiento de diversidad en DE	Prevención de pérdida de diversidad en alta dimensionalidad mediante partición del espacio, coevolución o uso de archivos de soluciones.
Submétodos:	
- *Multipoblación*	Subpoblaciones con estrategias independientes para promover exploración y explotación.
- *Archivos externos*	Almacenamiento de soluciones descartadas o fallidas para diversificar el cruce y la mutación.
Actualización de PSO	Modificación de reglas de actualización para evitar convergencia prematura y mejorar la exploración en alta dimensionalidad.
Submétodos:	
- *Reglas de actualización alternativas*	Reducción de la dependencia en global-best mediante aprendizajes sociales o mutaciones distribuidas.
- *Topologías dinámicas*	Uso de vecindarios dinámicos o estructuras de multipoblación para mejorar la diversidad.
Mantenimiento de diversidad en PSO	Mecanismos para equilibrar la exploración y explotación en grandes dimensiones.
Submétodos:	
- *Reinicialización parcial*	Reubicación de partículas en áreas de alta actividad para concentrar la búsqueda.
- *Muestreo basado en oposición*	Exploración en espacios opuestos para aumentar las probabilidades de mejora.
Partición del espacio en PSO	División del espacio de búsqueda en subregiones optimizadas por separado para mejorar el enfoque y evitar estancamientos.
Submétodos:	
- *Agrupamiento de dimensiones*	Actualización segmentada de dimensiones para evitar convergencia prematura.
- *Subenjambres*	Subenjambres independientes que comparten información de forma controlada para mantener la diversidad.

Tabla 3.: Resumen de operadores de muestreo y variación en DE y PSO para optimización global a gran escala.

Una vez terminada una exposición de donde se sitúan los algoritmos que vamos a utilizar,

discutiremos sus propiedades teóricas y realizaremos un estudio comparativo para comprobar si el agrupamiento diferencial de variables es efectivo en todos los casos. Llegaremos a la conclusión de que no se puede aplicar directamente la descomposición de variables con cualquier algoritmo. Los resultados de SHADE-ILS, demuestran que en algoritmos pensados para al alta dimensionalidad, aplicar descomposición no necesariamente mejora los resultados obtenidos, y es necesario realizar mejoras en este algoritmo si se quiere utilizar con problemas descompuestos. Esto último, se deja como trabajo futuro.

Presupesto

Aquí irá el presupuesto estimado del proyecto, basándose en el sueldo de un programador que deberíamos contratar para realizar el proyecto en función del número de horas y el coste de los servidores necesarios para realizar los cálculos de los algoritmos utlizados.

File: `preliminares/presupuesto.tex`

Parte I.

Parte Matemática

1. Preliminares

En esta sección se definen algunos conceptos matemáticos básicos, que se espera que se conozcan y no serán el objetivo de estudio de este trabajo, pero que son necesarios y se utilizarán durante el desarrollo de este trabajo.

1.1. Conceptos matemáticos fundamentales

Definición 1.1 (Derivada)**.** La derivada de una función $f : \mathbb{R} \to \mathbb{R}$ en un punto $x_0 \in \mathbb{R}$ se define como el límite, si existe:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

Este valor representa la pendiente de la recta tangente a la gráfica de f en el punto x_0.

Definición 1.2 (Derivada direccional)**.** Sea $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ una función diferenciable, y $\mathbf{u} = (u_1, \ldots, u_n)$ un vector de $\mathcal{U}$. La derivada direccional de f en la dirección $\mathbf{u}$, denotada como $D_{\mathbf{u}} f(x)$, está dada por:

$$D_{\mathbf{u}} f(x) = \sum_{i=1}^{n} \frac{\partial f(x)}{\partial x_i} u_i.$$

Definición 1.3 (Derivada parcial)**.** Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función. La derivada parcial de f con respecto a la variable x_i, evaluada en un punto $\mathbf{x}_0 = (x_1, \ldots, x_n)$, está dada por:

$$\frac{\partial f}{\partial x_i}(\mathbf{x}_0) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}.$$

Esto mide cómo cambia f al variar únicamente x_i, manteniendo las demás variables constantes. Es un caso particular de la derivada direccional, en la que el vector $\mathbf{u}$ es un vector de la base usual.

Definición 1.4 (Gradiente)**.** El gradiente de una función $f : \mathbb{R}^n \to \mathbb{R}$, denotado como ∇f, es el vector cuyas componentes son las derivadas parciales de f con respecto a cada variable:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix}.$$

El gradiente apunta en la dirección de mayor crecimiento de f.

Definición 1.5 (Hessiano)**.** La matriz hessiana de una función $f : \mathbb{R}^n \to \mathbb{R}$, denotada como $\nabla^2 f$, es la matriz cuadrada de segundo orden cuyas entradas son las derivadas parciales

segundas de f:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

La hessiana es simétrica si f es dos veces continuamente diferenciable.

Teorema 1.6 (Teorema de Taylor)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función continuamente diferenciable en un entorno abierto de $\mathbf{x}$. Entonces, para cualquier $\mathbf{p} \in \mathbb{R}^n$, existe $\theta \in (0,1)$ tal que*

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x}+\theta\mathbf{p})^\top \mathbf{p}.$$

Si además f es dos veces continuamente diferenciable, entonces

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \nabla^2 f(\mathbf{x}+\theta\mathbf{p})\mathbf{p}.$$

Teorema 1.7 (Teorema de Weierstrass)**.** *Sea $f : K \to \mathbb{R}$ una función continua definida sobre un conjunto $K \subseteq \mathbb{R}^n$ no vacío, cerrado y acotado. Entonces, f alcanza su máximo y su mínimo en K, es decir, existen puntos $\mathbf{x}_{max}, \mathbf{x}_{min} \in K$ tales que:*

$$f(\mathbf{x}_{min}) \leq f(\mathbf{x}) \leq f(\mathbf{x}_{max}), \quad \forall \mathbf{x} \in K.$$

Definición 1.8 (Integral de línea)**.** Sea $\mathbf{r}(t) = (x(t), y(t), z(t))$, $t \in [a,b]$, una curva diferenciable en $\mathbb{R}^3$, y sea $f : \mathbb{R}^3 \to \mathbb{R}$. La integral de línea de f a lo largo de $\mathbf{r}$ se define como:

$$\int_C f\, ds = \int_a^b f(\mathbf{r}(t))\|\mathbf{r}'(t)\|\, dt,$$

donde $\|\mathbf{r}'(t)\|$ es la norma de la derivada de $\mathbf{r}$, y ds representa un elemento diferencial de longitud de arco.

2. Optimización Numérica

La optimización numérica es una rama fundamental de las matemáticas aplicadas que se dedica al estudio y desarrollo de algoritmos para encontrar los valores óptimos (máximos o mínimos) de funciones, especialmente cuando estas son complejas y no pueden ser resueltas analíticamente [6]. Los problemas de optimización aparecen en diversas áreas como la ingeniería, la economía, la física y la inteligencia artificial.

Definición 2.1. Un **problema de optimización** consiste en encontrar el vector $\mathbf{x}^* \in \mathbb{R}^n$ que minimiza (o maximiza) una función objetivo $f : \mathbb{R}^n \to \mathbb{R}$, es decir:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}).$$

Los problemas de optimización pueden clasificarse en:

- **Optimización sin restricciones**: No existen limitaciones adicionales sobre las variables $\mathbf{x}$.
- **Optimización con restricciones**: Las variables $\mathbf{x}$ deben satisfacer ciertas condiciones, como igualdad o desigualdad.

2.1. Optimización Sin Restricciones

En la optimización sin restricciones, el objetivo es encontrar un punto donde la función objetivo alcanza su valor mínimo (o máximo) sin considerar limitaciones adicionales. Matemáticamente, esto implica resolver:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}).$$

En nuestro caso, consideraremos también problemas de optimización sin restricciones, a los que solo contienen restricciones del tipo:

$$a \leq x_i \leq b,$$

donde a y b son números reales, e $i = 1, 2, \ldots, n$, siendo n la dimensión del espacio. Es decir, las restricciones solo afectan a una única variable. Estas restricciones son útiles en la práctica, ya que no aumentan la complejidad del problema, solo limitan el espacio de búsqueda a una región acotada del espacio, que en la práctica es lo que nos interesa. Además, nos permiten asegurar que, si la función es continua, existe al menos una solución. Por el Teorema de Weierstrass 1.7, toda función continua definida en un compacto alcanza su máximo y su mínimo. Así que, si definimos una desigualdad del tipo:

$$a \leq x_i \leq b \quad \forall i,$$

tendremos un subconjunto compacto de $\mathbb{R}^n$ y podremos aspirar a encontrar el óptimo global de la función.

2.2. Optimización Con Restricciones

En muchos problemas prácticos, las variables están sujetas a restricciones. Estas pueden ser:

- **Restricciones de igualdad**: $h_i(\mathbf{x}) = 0$, $i = 1, \ldots, m$.
- **Restricciones de desigualdad**: $g_j(\mathbf{x}) \leq 0$, $j = 1, \ldots, p$.

El problema de optimización con restricciones se formula entonces como:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{sujeto a} \quad & h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, m, \\ & g_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, p. \end{aligned}$$

En nuestro caso, solo estamos interesados en estudiar la optimización numérica sin restricciones, así que nos centraremos en describir las técnicas más utilizadas en este campo sin tener en cuenta las técnicas utilizadas para la optimización con restricciones.

Nuestro objetivo ideal es encontrar un minimizador (para maximizador basta tomar $-f$) global de la función objetivo f, es decir, un punto donde la función alcanza su valor mínimo absoluto. Formalmente, definimos:

Definición 2.2. Un punto $\mathbf{x}^* \in \mathbb{R}^n$ es un **minimizador global** de f si

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

Sin embargo, encontrar el minimizador global puede ser difícil debido a la complejidad de f. Por lo tanto, a menudo nos conformamos con encontrar un minimizador local.

Definición 2.3. Un punto $\mathbf{x}^* \in \mathbb{R}^n$ es un **minimizador local** de f si existe un entorno N de $\mathbf{x}^*$ tal que

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in N.$$

Si la desigualdad es estricta para todos los $\mathbf{x} \neq \mathbf{x}^*$ en N, entonces $\mathbf{x}^*$ es un **minimizador local estricto**.

2.3. Condiciones para Óptimos Locales

Para identificar minimizadores locales, utilizamos condiciones basadas en las derivadas de f. Para ello nos será útil el teorema de Taylor 1.6, cuya demostración se puede encontrar en cualquier libro de cálculo básico.

2.3.1. Condiciones de Primer Orden

A continuación, describimos condiciones necesarias y suficientes para detectar mínimizadores locales de f.

Teorema 2.4 (Condiciones Necesarias de Primer Orden)**.** *Si $\mathbf{x}^*$ es un minimizador local de f y f es diferenciable en un entorno abierto de $\mathbf{x}^*$, entonces*

$$\nabla f(\mathbf{x}^*) = \mathbf{0}.$$

Demostración. Supongamos que $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$. Entonces, existe $\mathbf{p} = -\nabla f(\mathbf{x}^*)$ tal que, para $\alpha > 0$ suficientemente pequeño, se tiene $f(\mathbf{x}^* + \alpha\mathbf{p}) < f(\mathbf{x}^*)$, lo cual contradice que $\mathbf{x}^*$ es un minimizador local. □

2.3.2. Condiciones de Segundo Orden

Teorema 2.5 (Condiciones Necesarias de Segundo Orden)**.** *Si $\mathbf{x}^*$ es un minimizador local de f, f es dos veces diferenciable en un entorno abierto de $\mathbf{x}^*$, y $\nabla f(\mathbf{x}^*) = \mathbf{0}$, entonces*

$$\nabla^2 f(\mathbf{x}^*) \text{ es semidefinida positiva.}$$

Demostración. Si $\nabla^2 f(\mathbf{x}^*)$ no es semidefinida positiva, existe una dirección $\mathbf{p}$ tal que $\mathbf{p}^\top \nabla^2 f(\mathbf{x}^*)\mathbf{p} < 0$. Tomando un paso pequeño en esa dirección, podemos disminuir f, lo cual contradice que $\mathbf{x}^*$ es un minimizador local. □

Teorema 2.6 (Condiciones Suficientes de Segundo Orden)**.** *Si $\nabla f(\mathbf{x}^*) = \mathbf{0}$ y $\nabla^2 f(\mathbf{x}^*)$ es definida positiva, entonces $\mathbf{x}^*$ es un minimizador local estricto de f.*

Demostración. Dado que el hessiano es definida positiva, para $\mathbf{p} \neq \mathbf{0}$ suficientemente pequeño, el término cuadrático en el desarrollo de Taylor domina, y $f(\mathbf{x}^* + \mathbf{p}) > f(\mathbf{x}^*)$. □

2.4. Métodos de Búsqueda en Optimización

Existen dos estrategias principales para encontrar minimizadores locales en optimización sin restricciones: **métodos de búsqueda en línea** y **métodos de región de confianza**.

2.4.1. Métodos de Búsqueda en Línea

En los métodos de búsqueda en línea, cada iteración calcula una dirección de búsqueda $\mathbf{p}_k$ y luego decide cuánto avanzar en esa dirección. La iteración se define como:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

donde $\alpha_k > 0$ es la longitud del paso.

2.4.1.1. Selección de la Longitud del Paso

La elección de α_k es crucial. Idealmente, nos gustaría resolver el problema unidimensional:

$$\min_{\alpha > 0} f(\mathbf{x}_k + \alpha \mathbf{p}_k).$$

Sin embargo, resolverlo exactamente puede ser costoso, por lo que a menudo buscamos una aproximación que proporcione una reducción suficiente en f sin incurrir en un alto costo computacional.

2.4.1.2. Direcciones de Búsqueda

La mayoría de los algoritmos de búsqueda en línea requieren que $\mathbf{p}_k$ sea una dirección de descenso, es decir, que satisfaga:

$$\nabla f(\mathbf{x}_k)^\top \mathbf{p}_k < 0.$$

Un enfoque común es definir $\mathbf{p}_k$ como:

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k),$$

donde $\mathbf{B}_k$ es una matriz simétrica y no singular.

Descenso de Gradiente Si $\mathbf{B}_k = \mathbf{I}$, la identidad, entonces $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$, que es la dirección de descenso más pronunciado.

Método de Newton Si $\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k)$, entonces $\mathbf{p}_k$ es la dirección de Newton:

$$\mathbf{p}_k = -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k).$$

2.4.1.3. Condiciones de Wolfe

Para asegurar que la longitud del paso α_k proporcione una disminución suficiente en f, podemos utilizar las condiciones de Wolfe:

1. **Condición de Decrecimiento Suficiente**:

   $$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k,$$

   con $0 < c_1 < 1$.

2. **Condición de Curvatura**:

   $$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^\top \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k,$$

   con $c_1 < c_2 < 1$.

Estas condiciones garantizan que α_k no sea ni demasiado pequeño ni demasiado grande, proporcionando un equilibrio entre progreso y estabilidad.

Lema 2.7. *Sea $f : \mathbb{R}^n \to \mathbb{R}$ continuamente diferenciable, y $\mathbf{p}_k$ una dirección de descenso en $\mathbf{x}_k$. Entonces, si f está acotada inferiormente a lo largo de la dirección $\mathbf{p}_k$, existen intervalos de α_k que satisfacen las condiciones de Wolfe.*

Demostración. Dado que f está acotada inferiormente a lo largo de $\mathbf{p}_k$, la función unidimensional $f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ es también acotada inferiormente para $\alpha > 0$. Por otro lado, la línea $f(\mathbf{x}_k) + c_1 \alpha \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$, que representa la condición de decrecimiento suficiente 1, es no acotada inferiormente cuando $\alpha \to \infty$. Por lo tanto, debe existir al menos un punto $\alpha_0 > 0$ donde ambas gráficas se intersecten, es decir, tal que:

$$f(\mathbf{x}_k + \alpha_0 \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_0 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k.$$

Ahora, aplicando el teorema del valor medio, existe un $\bar{\alpha} \in (0, \alpha_0)$ tal que:

$$f(\mathbf{x}_k + \alpha_0 \mathbf{p}_k) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k + \bar{\alpha}\mathbf{p}_k)^\top (\alpha_0 \mathbf{p}_k).$$

Dividiendo por α_0 y combinando con la condición de decrecimiento suficiente 1, obtenemos:

$$\nabla f(\mathbf{x}_k + \bar{\alpha}\mathbf{p}_k)^\top \mathbf{p}_k \geq c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k,$$

donde $0 < c_1 < c_2 < 1$. Esto implica que también se satisface la condición de curvatura 2.

Dado que f es continuamente diferenciable, los valores de α_k que satisfacen ambas condiciones 1 y 2 forman un intervalo no vacío alrededor de α_0. Por lo tanto, existe un intervalo de α_k en el cual se satisfacen las condiciones de Wolfe. □

2.4.1.4. Algoritmo de Búsqueda de Longitud de Paso

Presentamos un algoritmo para encontrar α_k que satisface las condiciones de Wolfe.

Algorithm 1 Algoritmo de Búsqueda de Longitud de Paso

Inicializar $\alpha_0 = 0$, $\alpha_{\max} > 0$, elegir $\alpha_1 \in (0, \alpha_{\max})$.
$i \leftarrow 1$.
repeat
  Evaluar $f(\mathbf{x}_k + \alpha_i \mathbf{p}_k)$.
  if $f(\mathbf{x}_k + \alpha_i \mathbf{p}_k) > f(\mathbf{x}_k) + c_1 \alpha_i \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$ **ó** $[f(\mathbf{x}_k + \alpha_i \mathbf{p}_k) \geq f(\mathbf{x}_k + \alpha_{i-1} \mathbf{p}_k)]$ **y** $i > 1$ **then**
    Llamar a zoom(α_{i-1}, α_i) y **detener**.
  end if
  Evaluar $\nabla f(\mathbf{x}_k + \alpha_i \mathbf{p}_k)$.
  if $|\nabla f(\mathbf{x}_k + \alpha_i \mathbf{p}_k)^\top \mathbf{p}_k| \leq -c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$ **then**
    $\alpha_k \leftarrow \alpha_i$ y **detener**.
  end if
  if $\nabla f(\mathbf{x}_k + \alpha_i \mathbf{p}_k)^\top \mathbf{p}_k \geq 0$ **then**
    Llamar a zoom(α_i, α_{i-1}) y **detener**.
  end if
  Elegir $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$.
  $i \leftarrow i + 1$.
until Condición de terminación

Función Zoom La función zoom busca un α_k que satisfaga las condiciones de Wolfe dentro del intervalo $[\alpha_{\text{lo}}, \alpha_{\text{hi}}]$.

Algorithm 2 Función zoom($\alpha_{\text{lo}}, \alpha_{\text{hi}}$)

repeat
  Interpolar para encontrar α_j entre α_{lo} y α_{hi}.
  Evaluar $f(\mathbf{x}_k + \alpha_j \mathbf{p}_k)$.
  if $f(\mathbf{x}_k + \alpha_j \mathbf{p}_k) > f(\mathbf{x}_k) + c_1 \alpha_j \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$ **ó** $f(\mathbf{x}_k + \alpha_j \mathbf{p}_k) \geq f(\mathbf{x}_k + \alpha_{\text{lo}} \mathbf{p}_k)$ **then**
    $\alpha_{\text{hi}} \leftarrow \alpha_j$.
  else
    Evaluar $\nabla f(\mathbf{x}_k + \alpha_j \mathbf{p}_k)$.
    if $|\nabla f(\mathbf{x}_k + \alpha_j \mathbf{p}_k)^\top \mathbf{p}_k| \leq -c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{p}_k$ **then**
      $\alpha_k \leftarrow \alpha_j$ y **detener**.
    end if
    if $(\nabla f(\mathbf{x}_k + \alpha_j \mathbf{p}_k)^\top \mathbf{p}_k)(\alpha_{\text{hi}} - \alpha_{\text{lo}}) \geq 0$ **then**
      $\alpha_{\text{hi}} \leftarrow \alpha_{\text{lo}}$.
    end if
    $\alpha_{\text{lo}} \leftarrow \alpha_j$.
  end if
until Condición de terminación

2.5. Métodos de Región de Confianza

En los métodos de región de confianza, se construye un modelo cuadrático $m_k(\mathbf{p})$ que aproxima f cerca de $\mathbf{x}_k$:

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \frac{1}{2} \mathbf{p}^\top \mathbf{B}_k \mathbf{p},$$

donde $\mathbf{B}_k$ es una aproximación al hessiano. Se resuelve el subproblema:

$$\min_{\mathbf{p}} m_k(\mathbf{p}), \quad \text{sujeto a} \quad \|\mathbf{p}\| \leq \Delta_k,$$

donde Δ_k es el radio de la región de confianza.

2.6. Métodos Quasi-Newton

Los métodos quasi-Newton buscan aproximar el hessiano sin calcularlo directamente, utilizando únicamente evaluaciones del gradiente.

2.6.1. Actualización del Hessiano

Se basa en la condición de **secante**:

$$\mathbf{B}_{k+1} \mathbf{s}_k = \mathbf{y}_k,$$

donde:

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,$$
$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

2.6.2. El Algoritmo BFGS

El método BFGS, nombrado por Broyden, Fletcher, Goldfarb y Shanno, es uno de los algoritmos quasi-Newton más populares. Se basa en la construcción de un modelo cuadrático de la función objetivo en el punto actual $\mathbf{x}_k$:

$$m_k(\mathbf{p}) = f_k + \nabla f_k^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \mathbf{B}_k \mathbf{p},$$

donde $f_k = f(\mathbf{x}_k)$ y $\nabla f_k = \nabla f(\mathbf{x}_k)$. La matriz $\mathbf{B}_k$ es una aproximación simétrica y definida positiva del hessiano que se actualiza en cada iteración.

La dirección de búsqueda $\mathbf{p}_k$ se obtiene minimizando el modelo cuadrático:

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f_k.$$

El nuevo punto se calcula como:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

donde α_k es el tamaño de paso determinado mediante una búsqueda en línea que satisface las condiciones de Wolfe.

La actualización de $\mathbf{B}_k$ se realiza utilizando la siguiente fórmula, conocida como la fórmula de BFGS:

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{B}_k}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k}.$$

Esta fórmula garantiza que $\mathbf{B}_{k+1}$ sea simétrica y definida positiva si $\mathbf{B}_k$ es definida positiva y se satisface la condición de curvatura:

$$\mathbf{s}_k^\top \mathbf{y}_k > 0.$$

Algorithm 3 Algoritmo BFGS

1: Dado un punto inicial $\mathbf{x}_0$, una tolerancia $\epsilon > 0$, y una matriz inicial $\mathbf{B}_0$ definida positiva.
2: $k \leftarrow 0$.
3: **while** $\|\nabla f_k\| > \epsilon$ **do**
4: Calcular dirección de búsqueda: $\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f_k$.
5: Realizar búsqueda en línea para encontrar α_k que satisfaga las condiciones de Wolfe.
6: Actualizar el punto: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.
7: Calcular $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ y $\mathbf{y}_k = \nabla f_{k+1} - \nabla f_k$.
8: Actualizar $\mathbf{B}_{k+1}$ usando la fórmula de BFGS.
9: $k \leftarrow k + 1$.
10: **end while**

2.7. Optimización en alta dimensionalidad

Para problemas donde el número de variables es elevado, técnicas como BFGS se vuelven inmanejables debido a que requieren manipular y almacenar matrices densas de gran tamaño. Esto puede causar problemas de rendimiento o de memoria, especialmente en sistemas con recursos limitados. Para abordar estas limitaciones, se desarrollaron versiones adaptadas como el algoritmo L-BFGS, diseñado específicamente para optimización en alta dimensionalidad.

2.7.1. El Algoritmo L-BFGS

El algoritmo L-BFGS (*Limited-memory Broyden-Fletcher-Goldfarb-Shanno*) es una técnica eficiente para problemas de optimización de gran escala, donde almacenar y manipular la matriz hessiana completa es impráctico. Este método utiliza un número limitado de pares de vectores $\{\mathbf{s}_i, \mathbf{y}_i\}$ provenientes de las iteraciones más recientes. Estos pares capturan la información de curvatura necesaria para actualizar la dirección de búsqueda, mientras que la información más antigua se descarta para ahorrar memoria. Esto permite reducir significativamente los requerimientos de almacenamiento y cómputo, manteniendo una tasa de convergencia aceptable.

A continuación, se presenta el algoritmo L-BFGS:

Algorithm 4 Algoritmo L-BFGS

1: Elegir un punto inicial $\mathbf{x}_0$, un entero $m > 0$ y un criterio de convergencia.
2: Inicializar $k \leftarrow 0$.
3: **repeat**
4: Elegir la matriz inicial $\mathbf{H}_0^k$ (por ejemplo, usando la ecuación (2.1)).
5: Calcular la dirección de búsqueda $\mathbf{p}_k$ usando la *recursión de dos bucles* (ver Algoritmo 5).
6: Actualizar el punto: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, donde α_k satisface las condiciones de Wolfe.
7: Calcular los nuevos pares de vectores:

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \quad \mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

8: **if** $k \geq m$ **then**
9: Eliminar el par más antiguo $\{\mathbf{s}_{k-m}, \mathbf{y}_{k-m}\}$ de la memoria.
10: **end if**
11: Almacenar el nuevo par $\{\mathbf{s}_k, \mathbf{y}_k\}$.
12: Incrementar el contador: $k \leftarrow k + 1$.
13: **until** Se cumple el criterio de convergencia.

2.7.1.1. Recursión de dos bucles

El cálculo de la dirección de búsqueda $\mathbf{p}_k$ se realiza mediante la *recursión de dos bucles*, que calcula el producto $\mathbf{H}_k \nabla f_k$ sin construir explícitamente $\mathbf{H}_k$. Este procedimiento es esencial para mantener la eficiencia computacional del método. A continuación, se describe el proceso:

Algorithm 5 Recursión de dos bucles para L-BFGS

1: Dado ∇f_k, los pares $\{\mathbf{s}_i, \mathbf{y}_i\}$, y la matriz inicial $\mathbf{H}_0^k$.
2: Inicializar $\mathbf{q} \leftarrow \nabla f_k$.
3: **for** $i = k-1$ hasta $\max(k-m, 0)$ **do**
4: $\quad \alpha_i \leftarrow \frac{\mathbf{s}_i^\top \mathbf{q}}{\mathbf{y}_i^\top \mathbf{s}_i}$.
5: $\quad \mathbf{q} \leftarrow \mathbf{q} - \alpha_i \mathbf{y}_i$.
6: **end for**
7: $\mathbf{r} \leftarrow \mathbf{H}_0^k \mathbf{q}$.
8: **for** $i = \max(k-m, 0)$ hasta $k-1$ **do**
9: $\quad \beta \leftarrow \frac{\mathbf{y}_i^\top \mathbf{r}}{\mathbf{y}_i^\top \mathbf{s}_i}$.
10: $\quad \mathbf{r} \leftarrow \mathbf{r} + \mathbf{s}_i(\alpha_i - \beta)$.
11: **end for**
12: Devolver $\mathbf{p}_k \leftarrow -\mathbf{r}$.

2.7.1.2. Elección de la matriz inicial $\mathbf{H}_0^k$

La matriz inicial $\mathbf{H}_0^k$ se utiliza para escalar la dirección de búsqueda y mejorar la convergencia del algoritmo. Una elección común es:

$$\mathbf{H}_0^k = \gamma_k \mathbf{I}, \tag{2.1}$$

donde

$$\gamma_k = \frac{\mathbf{s}_{k-1}^\top \mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^\top \mathbf{y}_{k-1}}.$$

Esta estrategia permite capturar información de curvatura reciente, asegurando que la dirección de búsqueda esté bien escalada.

2.7.2. Ventajas del L-BFGS

- **Memoria limitada:** El algoritmo almacena solo los últimos m pares $\{\mathbf{s}_i, \mathbf{y}_i\}$, lo que reduce significativamente los requisitos de almacenamiento comparado con BFGS estándar.
- **Eficiencia computacional:** La recursión de dos bucles tiene un costo de $4mn$ operaciones, donde n es el número de variables y m el número de pares almacenados.
- **Aplicaciones prácticas:** Es ampliamente utilizado en problemas de optimización de alta dimensionalidad, como el ajuste de parámetros en modelos de aprendizaje automático y simulaciones numéricas.

3. Agrupamiento de variables

En esta sección, estudiaremos los fundamentos teóricos del agrupamiento diferencial [7] [11] [14] [15] , una técnica de agrupamiento automático de variables que permite descomponer un problema en subproblemas menores. Esta metodología pretende dividir un conjunto de variables acorde a la interdependencia que se establece entre ellas cuando se trata de optimizar una función objetivo.

Al igual que en el análisis clúster, que agrupa conjuntos de datos en grupos de forma que en cada uno, los datos sean lo más parecidos posibles y distintos del resto de clústers, el objetivo de esta técnica es separar las variables en conjuntos, de forma que cada uno sea independiente del resto y dentro de cada conjunto ninguna variable sea independiente. La principal diferencia radica en que en el análisis clúster agrupamos datos acorde al valor de las variables y en el agrupamiento diferencial lo que agrupamos son las variables acorde a la dependencia que existe entre ellas.

Las dos principales ventajas del agrupamiento diferencial con respecto a otras técnicas de agrupamiento, como el agrupamiento aleatorio, o el agrupamiento en k-componetes son:

- **No requiere de parámetros:** el número y tamaño de los grupos se decide acorde a las interacciones detectadas en las variables, no es necesario fijar el numero de grupos ni el tamaño de estos previamente.
- **Tiene una estrategia:** utiliza pequeñas perturbaciones en las variables para detectar si dos variables interactúan y agruparlas.

A continuación, se exponen las definiciones y teoremas necesarios para entender el agrupamiento diferencial desde un punto de vista teórico. En la segunda parte de este TFG, se implementarán distintas variantes de esta técnica para probar su efectividad a la hora de hibridarlas con algoritmos que permitan optimizar una función objetivo.

3.1. Definiciones

Definición 3.1. Una función $f(x_1, \ldots, x_n)$ es separable si y solo si:

$$\arg \min_{(x_1,\ldots,x_n)} f(x_1, \ldots, x_n) = \left(\arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_n} f(\ldots, x_n) \right)$$

y no separable en otro caso.

Es decir, podemos encontrar el óptimo de esa función optimizando en cada dimensión por separado.

Definición 3.2. Una función $f(x)$ se dice parcialmente separable con m componentes si y solo si:

$$\arg\min_{x} f(x) = \left(\arg\min_{x_1} f(x_1, \ldots), \ldots, \arg\min_{x_m} f(\ldots, x_m) \right)$$

donde $x = (x_1, \ldots, x_n)$ es un vector de decisión de n dimensiones, $x_1, \ldots, x_n$ son subvectores disjuntos de x y $2 \leq m \leq n$.

La definición 3.2 difiere de 3.1 en que ahora los vectores x_i no son unidimensionales, es decir, podemos encontrar el óptimo optimizando cada subvector por separado, pero cada subvector puede requerir optimizar un conjunto de variables en vez de una única variable cada vez. La definición 3.1 es un caso particular de 3.2 cuando $n = m$.

Definición 3.3. Una función es parcialmente aditivamente separable si es de la siguiente forma:

$$f(x) = \sum_{i=1}^{m} f_i(x_i), \quad m > 1$$

donde $f_i(\cdot)$ son subfunciones que solo dependen de las variables que forman cada vector x_i y x y x_i se definen como en 3.2. En el caso en el que $n = m$, la función se dice totalmente aditivamente separable.

La definición 3.3 es un caso especial de la definición 3.2. Un ejemplo de este tipo de funciones puede ser la función $f(x) = x_1^2 + x_2^2$. Es claro que $f(x) = f_1(x_1) + f_2(x_2)$ con $f_1(x_1) = x_1^2$ y $f_2(x_2) = x_2^2$.

Definición 3.4. Se dice que dos variables x e y interactúan, si no pueden ser optimizadas de forma independiente. Lo denotaremos por $x \leftrightarrow y$.

Introducimos ahora definiciones que serán útiles para detectar variables que interactúan entre sí. La interacción entre variables es otro enfoque de la separabilidad de funciones, pero que nos será útil a la hora de diseñar el algoritmo de agrupamiento diferencial recursivo.

Definición 3.5. Sea $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ una función diferenciable. Las variables de decisión x_i y x_j interactúan si existe una solución candidata $\mathbf{x}^*$ tal que

$$\frac{\partial^2 f(\mathbf{x}^*)}{\partial x_i \partial x_j} \neq 0,$$

y diremos que interactúan condicionalmente si

$$\frac{\partial^2 f(\mathbf{x}^*)}{\partial x_i \partial x_j} = 0,$$

y existe un conjunto de variables de decisión $\{x_{k1}, \ldots, x_{kt}\} \subset X$, tal que $x_i \leftrightarrow x_{k1} \leftrightarrow \ldots \leftrightarrow x_{kt} \leftrightarrow x_j$.

Las variables de decisión x_i y x_j son independientes si para cualquier solución candidata $\mathbf{x}^*$, se cumple la ecuación anterior y no existe un conjunto de variables de decisión $\{x_{k1}, \ldots, x_{kt}\} \subset X$, tal que $x_i \leftrightarrow x_{k1} \leftrightarrow \ldots \leftrightarrow x_{kt} \leftrightarrow x_j$.

3.2. Teoremas

En esta sección se presentan los teoremas necesarios para comprender el agrupamiento diferencial.

3.2.1. Agrupamiento diferencial

Teorema 3.6. *Sea $f(\mathbf{x})$ una función (parcialemente) aditivamente separable. Para todo $a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0$, si se cumple la siguiente condición:*

$$\Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_2} \tag{3.1}$$

entonces x_p y x_q son no separables, donde

$$\Delta_{\delta,x_p}[f](\mathbf{x}) = f(\ldots, x_p + \delta, \ldots) - f(\ldots, x_p, \ldots)$$

se refiere a la diferencia hacia adelante de f con respecto a la variable x_p con intervalo δ.

El Teorema 3.6 sencillamente nos dice que, dada una función aditivamente separable $f(\mathbf{x})$, dos variables x_p y x_q interactúan si la diferencia hacia adelante evaluada con dos valores diferentes para x_q produce resultados diferentes.

Para probar el teorema es suficiente demostrar su contrarrecíproco, que establece que si dos variables x_p y x_q son separables, entonces la diferencia hacia adelante evaluada con dos valores diferentes para x_q produce el mismo resultado.

Lema 3.7. *Si $f(\mathbf{x})$ es aditivamente separable, entonces para cualquier $x_p \in \mathbf{x}$ tenemos*

$$\frac{\partial f(\mathbf{x})}{\partial x_p} = \frac{\partial f_i(\mathbf{x}_i)}{\partial x_p}, \quad \forall x_p \in \mathbf{x}_i.$$

Demostración. Dado que $f(\mathbf{x})$ es aditivamente separable, tenemos

$$\frac{\partial f(\mathbf{x})}{\partial x_p} = \frac{\partial}{\partial x_p} \sum_{i=1}^{m} f_i(\mathbf{x}_i) = \frac{\partial f_1(\mathbf{x}_1)}{\partial x_p} + \cdots + \frac{\partial f_m(\mathbf{x}_m)}{\partial x_p}, \quad \forall x_p \in \mathbf{x}_i$$

donde $\mathbf{x}_1, \ldots, \mathbf{x}_m$ son vectores de decisión mutuamente excluyentes. Por lo tanto,

$$\frac{\partial f(\mathbf{x}_j)}{\partial x_p} = 0, \quad \forall j \neq i.$$

Así,

$$\frac{\partial f(\mathbf{x})}{\partial x_p} = \frac{\partial f_i(\mathbf{x}_i)}{\partial x_p}, \quad \forall x_p \in \mathbf{x}_i.$$

□

Demostración del Teorema 3.6. Según el Lema 3.7,

$$\frac{\partial f(\mathbf{x})}{\partial x_p} = \frac{\partial f_i(\mathbf{x}_i)}{\partial x_p}, \quad \forall x_p \in \mathbf{x}_i.$$

Entonces, para todo $x_q \notin \mathbf{x}_i$ tenemos

$$\left.\frac{\partial f(\mathbf{x})}{\partial x_p}\right|_{x_q=b_1} = \left.\frac{\partial f(\mathbf{x})}{\partial x_p}\right|_{x_q=b_2} = \frac{\partial f_i(\mathbf{x}_i)}{\partial x_p}, \quad \forall b_1 \neq b_2.$$

$$\int_a^{a+\delta} \frac{\partial f(\mathbf{x})}{\partial x_p}\, dx_p \Big|_{x_q=b_1} = \int_a^{a+\delta} \frac{\partial f(\mathbf{x})}{\partial x_p}\, dx_p \Big|_{x_q=b_2}$$

$$\Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b1} = \Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b2} \quad \forall a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0.$$

□

El Teorema 3.6 es el que nos servirá en la segunda parte para diseñar el algoritmo de agrupamiento diferencial.

3.2.2. Agrupamiento diferencial recursivo

A continuación se presentan los teoremas y proposiciones necesarios para el agrupamiento diferencial recursivo.

Notación: Sea X el conjunto de variables de decisión $\{x_1, \ldots, x_n\}$ y U_X el conjunto de vectores unitarios en el espacio de decisión $\mathbb{R}^n$. Sea X_1 un subconjunto de variables de decisión $X_1 \subset X$ y U_{X_1} un subconjunto de U_X tal que para cualquier vector unitario $\mathbf{u} = (u_1, \ldots, u_n) \in U_{X_1}$, tenemos $u_i = 0$ si $x_i \notin X_1$.

Proposición 3.8. *Sea $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ una función diferenciable; $X_1 \subset X$ y $X_2 \subset X$ dos subconjuntos mutuamente excluyentes de variables de decisión: $X_1 \cap X_2 = \emptyset$. Si existen dos vectores unitarios $\mathbf{u}_1 \in U_{X_1}$ y $\mathbf{u}_2 \in U_{X_2}$, y una solución candidata $\mathbf{x}^*$ en el espacio de decisión tal que*

$$D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}^*) \neq 0$$

entonces hay alguna interacción entre las variables de decisión en X_1 y X_2.

Demostración. Sin pérdida de generalidad, asumimos que $X_1 = \{x_{1,1}, \ldots, x_{1,p}\}$, $X_2 = \{x_{2,1}, \ldots, x_{2,q}\}$, donde p y q son el número de variables de decisión en X_1 y X_2, respectivamente; $\mathbf{u}_1 = (u_1^1, \ldots, u_1^n)$ y $\mathbf{u}_2 = (u_2^1, \ldots, u_2^n)$. Según la derivada direccional,

$$D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} u_1^i u_2^j.$$

Como $\mathbf{u}_1$ y $\mathbf{u}_2$ son dos vectores unitarios de U_{X_1} y U_{X_2}, respectivamente, podemos obtener que

$$u_1^i = 0, \text{ si } x_i \notin X_1,$$

$$u_2^j = 0, \text{ si } x_j \notin X_2.$$

Por lo tanto,

$$D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}) = \sum_{i=1}^{p} \sum_{j=1}^{q} \frac{\partial^2 f(\mathbf{x})}{\partial x_{1,i} \partial x_{2,j}} u_1^{1,i} u_2^{2,j}.$$

Si se cumple (3.1),

$$\sum_{i=1}^{p} \sum_{j=1}^{q} \frac{\partial^2 f(\mathbf{x}^*)}{\partial x_{1,i} \partial x_{2,j}} u_1^{1,i} u_2^{2,j} \neq 0.$$

Por lo tanto, existe al menos un par (i, j), tal que

$$\frac{\partial^2 f(\mathbf{x}^*)}{\partial x_{1,i} \partial x_{2,j}} \neq 0.$$

Basado en la Definición 3.5, al menos un par de variables de decisión $x_{1,i} \in X_1$ y $x_{2,j} \in X_2$ interactúan. □

Corolario 3.9. *Sea $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ una función objetivo; $X_1 \subset X$ y $X_2 \subset X$ dos subconjuntos mutuamente excluyentes de variables de decisión: $X_1 \cap X_2 = \emptyset$. Si existen dos vectores unitarios $\mathbf{u}_1 \in U_{X_1}$ y $\mathbf{u}_2 \in U_{X_2}$, dos números reales $l_1, l_2 > 0$, y una solución candidata $\mathbf{x}^*$ en el espacio de decisión, tal que*

$$f(\mathbf{x}^* + l_1\mathbf{u}_1 + l_2\mathbf{u}_2) - f(\mathbf{x}^* + l_2\mathbf{u}_2) \neq f(\mathbf{x}^* + l_1\mathbf{u}_1) - f(\mathbf{x}^*) \tag{3.2}$$

entonces hay alguna interacción entre las variables de decisión en X_1 y X_2.

Demostración. Con la Proposición 3.8, solo necesitamos probar la siguiente afirmación.

Afirmación 1: Si existen dos vectores unitarios $\mathbf{u}_1 \in U_{X_1}$ y $\mathbf{u}_2 \in U_{X_2}$, dos números reales $l_1, l_2 > 0$, y una solución candidata $\mathbf{x}^*$ en el espacio de decisión, tal que (3.2) se cumple, entonces (3.1) es verdadero.

Es equivalente probar su contrarecíproco.

Afirmación 2: Si para cualquier par de vectores unitarios $\mathbf{u}_1 \in U_{X_1}$ y $\mathbf{u}_2 \in U_{X_2}$, y para cualquier solución candidata $\mathbf{x}^*$ en el espacio de decisión, se cumple la siguiente condición:

$$D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}^*) = 0 \tag{3.3}$$

entonces

$$f(\mathbf{x}^* + l_1\mathbf{u}_1 + l_2\mathbf{u}_2) - f(\mathbf{x}^* + l_2\mathbf{u}_2) = f(\mathbf{x}^* + l_1\mathbf{u}_1) - f(\mathbf{x}^*)$$

para cualquier $l_1, l_2 > 0$.

Sea $A_2(\mathbf{x}^*)$ cualquier punto en $\mathbb{R}^n$, y B_2 sea $\mathbf{x}^* + l_2\mathbf{u}_2$, donde $\mathbf{u}_2$ es cualquier vector en U_{X_2} y l_2 es cualquier número real positivo. Sea C_2 cualquier punto en el segmento A_2B_2. Por lo tanto, la longitud del segmento A_2B_2 es l_2, y la coordenada de $C_2(\mathbf{x})$ puede ser determinada de manera única por la longitud del segmento $A_2C_2(s_2)$: $\mathbf{x}(s_2) = \mathbf{x}^* + s_2\mathbf{u}_2$, $s_2 \in [0, l_2]$. Si (3.3) se cumple para cualquier solución candidata en el espacio de decisión, entonces

$$D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}) = 0. \tag{3.4}$$

Como $D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}) = D_{\mathbf{u}_2} D_{\mathbf{u}_1} f(\mathbf{x})$, integrando ambos lados de (3.4) a lo largo del segmento A_2B_2, obtenemos

$$\int_0^{l_2} D_{\mathbf{u}_1} D_{\mathbf{u}_2} f(\mathbf{x}) ds_2 = \int_0^{l_2} D_{\mathbf{u}_2} D_{\mathbf{u}_1} f(\mathbf{x}) ds_2 = 0.$$

Como

$$\int_0^{l_2} D_{\mathbf{u}_2}(D_{\mathbf{u}_1} f(\mathbf{x}(s_2))) ds_2 = D_{\mathbf{u}_1} f(\mathbf{x}(s_2))\big|_{s_2=0}^{s_2=l_2},$$

entonces,

$$D_{\mathbf{u}_1} f(\mathbf{x}(s_2))\big|_{s_2=0}^{s_2=l_2} = 0$$

y

$$D_{\mathbf{u}_1} f(\mathbf{x}^* + l_2\mathbf{u}_2) - D_{\mathbf{u}_1} f(\mathbf{x}^*) = 0.$$

Como $A_2(\mathbf{x}^*)$ es cualquier punto en $\mathbb{R}^n$, entonces

$$D_{\mathbf{u}_1} f(\mathbf{x} + l_2\mathbf{u}_2) = D_{\mathbf{u}_1} f(\mathbf{x}). \tag{3.5}$$

Sea $A_1(\mathbf{x}^*)$ cualquier punto en $\mathbb{R}^n$, y B_1 sea $\mathbf{x}^* + l_1\mathbf{u}_1$, donde $\mathbf{u}_1$ es cualquier vector en U_{X_1} y l_1 es cualquier número real positivo. Sea C_1 cualquier punto en el segmento A_1B_1. Por lo tanto, la longitud del segmento A_1B_1 es l_1, y la coordenada de $C_1(\mathbf{x})$ puede ser determinada de manera única por la longitud del segmento $A_1C_1(s_1)$: $\mathbf{x}(s_1) = \mathbf{x}^* + s_1\mathbf{u}_1$, $s_1 \in [0, l_1]$. De manera similar, integrando ambos lados de (3.5) a lo largo del segmento A_1B_1, obtenemos

$$\int_0^{l_1} D_{\mathbf{u}_1} f(\mathbf{x}(s_1) + l_2\mathbf{u}_2)ds_1 = \int_0^{l_1} D_{\mathbf{u}_1} f(\mathbf{x}(s_1))ds_1.$$

Por lo tanto,

$$f(\mathbf{x}^* + l_1\mathbf{u}_1 + l_2\mathbf{u}_2) - f(\mathbf{x}^* + l_2\mathbf{u}_2) = f(\mathbf{x}^* + l_1\mathbf{u}_1) - f(\mathbf{x}^*).$$

Así, la Afirmación 2 queda probada, y la Afirmación 1 y el Corolario 3.9 son verdaderos. □

Supongamos que X_1 y X_2 son dos subconjuntos mutuamente exclusivos de variables. Si X_1 está relacionado con X_2, el método RDG divide X_2 en dos subconjuntos de igual tamaño y mutuamente exclusivos, X_3 y X_4.

Definimos:

$$\Delta_1 = f(x^* + l_1u_1 + l_2(u_3 + u_4)) - f(x^* + l_2(u_3 + u_4)),$$
$$\Delta_2 = f(x^* + l_1u_1) - f(x^*).$$

Proposición 3.10. *Si $(\Delta_1 - \Delta_2) = (\Delta_1' - \Delta_2')$, entonces X_1 no está relacionado con X_4; de lo contrario, X_1 está relacionado con X_4.*

Demostración. Si $(\Delta_1 - \Delta_2) = (\Delta_1' - \Delta_2')$, dado que $\Delta_2 = \Delta_2'$, es evidente que $\Delta_1 = \Delta_1'$:

$$f(x^* + l_1u_1 + l_2(u_3 + u_4)) - f(x^* + l_2(u_3 + u_4)) = f(x^* + l_1u_1 + l_2u_3) - f(x^* + l_2u_3). \quad (3.6)$$

Sea $x' = x^* + l_2u_3$. Entonces, la ecuación (3.6) se convierte en:

$$f(x' + l_1u_1 + l_2u_4) - f(x' + l_2u_4) = f(x' + l_1u_1) - f(x'). \quad (3.7)$$

La ecuación (3.7) indica que X_1 no está relacionado con X_4.

Si $(\Delta_1 - \Delta_2) \neq (\Delta_1' - \Delta_2')$, dado que $\Delta_2 = \Delta_2'$, es evidente que $\Delta_1 \neq \Delta_1'$:

$$f(x^* + l_1u_1 + l_2(u_3 + u_4)) - f(x^* + l_2(u_3 + u_4)) \neq f(x^* + l_1u_1 + l_2u_3) - f(x^* + l_2u_3). \quad (3.8)$$

Sea $x' = x^* + l_2u_3$. Entonces, la ecuación (3.8) se convierte en:

$$f(x' + l_1u_1 + l_2u_4) - f(x' + l_2u_4) \neq f(x' + l_1u_1) - f(x').$$

Esto implica que X_1 está relacionado con X_4.

□

4. Tests estadísticos

En el campo de la optimización, y en particular en el de los algoritmos genéticos y otros métodos estocásticos, la comparación de algoritmos es una tarea fundamental para determinar qué enfoque es más efectivo bajo ciertas condiciones. Sin embargo, realizar comparaciones significativas no es una tarea trivial, ya que los resultados de los algoritmos suelen estar sujetos a variabilidad, tanto por la naturaleza estocástica de muchos métodos como por las características particulares de los problemas analizados. Para abordar este desafío, es indispensable contar con herramientas estadísticas que permitan evaluar rigurosamente el desempeño de los algoritmos. Para ello nos basamos en los estudios realizados en [1] y [3].

Los tests estadísticos se utilizan para determinar si las diferencias observadas entre algoritmos son estadísticamente significativas o simplemente producto del azar. Entre los más comunes, se encuentran los tests paramétricos, como el *t-test* y el análisis de varianza (ANOVA), que asumen que los datos siguen distribuciones normales y presentan varianzas homogéneas. Estas herramientas son útiles para comparaciones simples y cuando se cumplen los supuestos necesarios. Sin embargo, en muchos escenarios, estas condiciones no se verifican, especialmente en el análisis de múltiples problemas o cuando las muestras son pequeñas.

Esto es particularmente problemático en algoritmos estocásticos, donde las distribuciones de los resultados pueden ser complejas o desconocidas. En este contexto, los tests no paramétricos han ganado popularidad debido a su menor sensibilidad a los supuestos sobre la distribución de los datos. Métodos como el test de Wilcoxon, el test de Friedman y el procedimiento *post-hoc* de Nemenyi son ampliamente utilizados en la comparación de algoritmos. Estas técnicas permiten analizar conjuntos de datos más diversos, evaluando múltiples algoritmos en diferentes problemas sin necesidad de cumplir estrictos requisitos estadísticos.

El uso de estos tests no solo mejora la calidad de las comparaciones experimentales, sino que también permite establecer conclusiones más robustas y generalizables sobre el desempeño relativo de los algoritmos. Su integración en los análisis experimentales es indispensable para avanzar hacia una evaluación científica rigurosa y confiable en el campo de la optimización y la inteligencia computacional.

En esta sección, exploraremos los tests no paramétricos más utilizados, dividiendo su análisis en dos categorías principales: tests de comparación por parejas, que permiten comparar directamente dos algoritmos, y tests de comparación múltiple, diseñados para evaluar simultáneamente tres o más algoritmos en múltiples problemas. Estas herramientas proporcionan un marco sólido para realizar análisis estadísticos significativos en condiciones donde los supuestos de los tests paramétricos no se cumplen.

4.1. Comparaciones por pares

Las comparaciones por pares son uno de los métodos estadísticos más básicos utilizados en estudios experimentales para comparar el rendimiento de dos algoritmos sobre un conjunto común de problemas. En un análisis que abarca múltiples problemas, se requiere un valor

para cada combinación algoritmo/problema, que usualmente es un promedio de varias ejecuciones.

En esta sección, presentamos dos procedimientos para realizar comparaciones por pares:

- La *prueba de signos* (Sección 4.1.1), un método simple y rápido que ofrece una primera aproximación, aunque con potencia estadística limitada.
- La *prueba de rangos con signo de Wilcoxon* (Sección 4.1.2), un enfoque no paramétrico más robusto y confiable para detectar diferencias significativas entre dos algoritmos.

4.1.1. Prueba de signos

La prueba de signos es una forma sencilla y popular de comparar el rendimiento global de dos algoritmos. Consiste en contar el número de problemas en los que un algoritmo supera al otro. Este conteo se utiliza en una prueba binomial bilateral conocida como *prueba de signos*. Bajo la hipótesis nula de equivalencia entre los algoritmos, se espera que cada uno gane aproximadamente en la mitad de los problemas ($n/2$).

El número de victorias sigue una distribución binomial. Para valores grandes de n, esta distribución se aproxima a una distribución normal con media $\mu = n/2$ y desviación estándar $\sigma = \sqrt{n/2}$. La estadística de prueba z se calcula como:

$$z = \frac{W - n/2}{\sqrt{n/2}},$$

donde W es el número de victorias de uno de los algoritmos. Si $|z|$ es mayor que 1.96 (para un nivel de significancia $\alpha = 0.05$), se rechaza la hipótesis nula y se concluye que existe una diferencia significativa entre los algoritmos.

Es importante no descontar los empates al aplicar esta prueba, ya que apoyan la hipótesis nula. En caso de empates, se dividen equitativamente entre ambos algoritmos. Si hay un número impar de empates, uno de ellos se ignora.

4.1.2. Prueba de rangos con signo de Wilcoxon

La prueba de rangos con signo de Wilcoxon es un método no paramétrico utilizado para determinar si existen diferencias significativas entre las medianas de dos muestras relacionadas. Es análoga al *t-test* pareado en procedimientos estadísticos no paramétricos y es especialmente útil cuando no se puede asumir normalidad en los datos.

El procedimiento es el siguiente:

1. Calcular las diferencias d_i entre los resultados de los dos algoritmos en cada problema i:

   $$d_i = x_{i1} - x_{i2},$$

   donde x_{i1} y x_{i2} son los resultados del primer y segundo algoritmo en el problema i, respectivamente. Si los puntajes están en diferentes escalas, pueden normalizarse al intervalo [0,1] para evitar priorizar algún problema.

2. Excluir las diferencias que sean cero ($d_i = 0$), ya que no aportan información sobre la dirección de la diferencia.

3. Ordenar las diferencias d_i según su valor absoluto y asignar rangos R_i, comenzando con 1 para la diferencia más pequeña. En caso de empates, se asignan rangos promedio.

4. Asignar a cada rango el signo de la diferencia correspondiente. Es decir, si $d_i > 0$, el rango R_i es positivo; si $d_i < 0$, el rango R_i es negativo.

5. Calcular las sumas de los rangos positivos y negativos:

$$R^+ = \sum_{d_i>0} R_i + \frac{1}{2} \sum_{d_i=0} R_i,$$

$$R^- = \sum_{d_i<0} R_i + \frac{1}{2} \sum_{d_i=0} R_i.$$

6. Determinar el estadístico de prueba T, que es el menor de R^+ y R^-:

$$T = \min(R^+, R^-).$$

7. Comparar T con el valor crítico de la distribución de Wilcoxon para el tamaño de muestra n. Si T es menor o igual al valor crítico, se rechaza la hipótesis nula, indicando una diferencia significativa entre los algoritmos.

La prueba de Wilcoxon es más sensible que el *t-test* pareado y no requiere la suposición de normalidad. Además, es menos afectada por valores atípicos, ya que los rangos reducen el impacto de observaciones extremas. Es importante no redondear las diferencias a pocos decimales, ya que esto puede disminuir la potencia de la prueba al aumentar el número de empates.

4.2. Comparaciones múltiples con un método de control

En muchas situaciones experimentales, es necesario comparar el rendimiento de varios algoritmos simultáneamente, especialmente cuando se desea evaluar un nuevo método frente a múltiples alternativas. Sin embargo, realizar múltiples comparaciones por pares incrementa el riesgo de cometer errores de Tipo I (rechazar incorrectamente la hipótesis nula) debido al acumulado de errores en cada prueba individual. Este fenómeno se conoce como la tasa de error familiar (*Family-Wise Error Rate*, FWER).

Si se comparan k algoritmos y se realiza cada prueba con un nivel de significancia α, la probabilidad de no cometer un error de Tipo I en una única comparación es $(1 - \alpha)$. Por tanto, la probabilidad de no cometer un error de Tipo I en todas las $k - 1$ comparaciones es $(1 - \alpha)^{k-1}$. De esta forma, la probabilidad de cometer al menos un error de Tipo I es:

$$1 - (1 - \alpha)^{k-1}.$$

Por ejemplo, si $\alpha = 0.05$ y $k = 9$, esta probabilidad es aproximadamente 0.33, lo cual es bastante alto.

Para abordar este problema, se utilizan pruebas estadísticas diseñadas para comparaciones múltiples con un método de control. En esta sección, se describen varios métodos adecuados para este propósito:

- La *prueba de Friedman* y sus extensiones (Sección 4.2.1).
- Procedimientos *post-hoc* para identificar diferencias específicas entre el método de control y los demás algoritmos (Sección 4.2.2).

4.2.1. Prueba de Friedman y extensiones

La prueba de Friedman es un test no paramétrico para comparar más de dos muestras relacionadas. Evalúa si existen diferencias significativas en las medianas de k algoritmos evaluados sobre n problemas.

El procedimiento es el siguiente:

1. Para cada problema i, asignar rangos r_{ij} a los algoritmos, donde r_{ij} es el rango del algoritmo j en el problema i. El mejor algoritmo recibe el rango 1, el siguiente mejor rango 2, y así sucesivamente. En caso de empates, se asignan rangos promedio.
2. Calcular el rango promedio R_j de cada algoritmo:
$$R_j = \frac{1}{n} \sum_{i=1}^{n} r_{ij}.$$
3. Calcular la estadística de Friedman:
$$\chi_F^2 = \frac{12n}{k(k+1)} \left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right).$$
4. Bajo la hipótesis nula de que todos los algoritmos tienen el mismo rendimiento, χ_F^2 sigue una distribución χ^2 con $k-1$ grados de libertad. Si el valor calculado es mayor que el valor crítico de la distribución, se rechaza la hipótesis nula.

Para ajustar la conservaduría de la prueba de Friedman, Iman y Davenport propusieron una modificación que utiliza una distribución F:

$$F_F = \frac{(n-1)\chi_F^2}{n(k-1) - \chi_F^2},$$

donde F_F sigue una distribución F con $k-1$ y $(k-1)(n-1)$ grados de libertad.

4.2.1.1. Prueba de rangos alineados de Friedman

La prueba de rangos alineados de Friedman mejora el enfoque anterior al considerar las diferencias en el rendimiento absoluto entre los algoritmos. El procedimiento es el siguiente:

1. Para cada problema i, calcular el valor central (por ejemplo, la mediana) de los resultados de todos los algoritmos, denotado como M_i.
2. Calcular las diferencias alineadas d_{ij} entre el resultado del algoritmo j en el problema i y el valor central M_i:
$$d_{ij} = x_{ij} - M_i.$$

3. Ordenar todos los valores d_{ij} (para todos los algoritmos y problemas) de menor a mayor y asignar rangos R_{ij}. En este caso, los rangos se asignan considerando todos los $k \times n$ valores conjuntamente.

4. Calcular el rango promedio alineado $\tilde{R}_j$ para cada algoritmo:

$$\tilde{R}_j = \frac{1}{n} \sum_{i=1}^{n} R_{ij}.$$

5. Calcular la estadística de Friedman alineada:

$$\chi^2_{F_{aligned}} = \frac{12}{k(k+1)} \left(\sum_{j=1}^{k} \tilde{R}_j^2 - \frac{k(k+1)^2}{4} \right).$$

6. Comparar $\chi^2_{F_{aligned}}$ con el valor crítico de la distribución χ^2 con $k-1$ grados de libertad.

Esta prueba es más sensible que la prueba de Friedman estándar, ya que tiene en cuenta las magnitudes de las diferencias entre los algoritmos.

4.2.1.2. Prueba de Quade

La prueba de Quade introduce ponderaciones basadas en la dificultad relativa de los problemas. El procedimiento es:

1. Calcular el rango r_{ij} de los algoritmos en cada problema i, como en la prueba de Friedman.

2. Para cada problema i, calcular el rango Q_i basado en la amplitud de las diferencias en ese problema. Esto se hace calculando la diferencia entre el mejor y el peor resultado en el problema i, y luego asignando rangos a los problemas según estas diferencias (el problema con la menor diferencia recibe el rango 1).

3. Calcular los estadísticos S_{ij}:

$$S_{ij} = Q_i \left(r_{ij} - \frac{k+1}{2} \right).$$

4. Calcular la suma S_j para cada algoritmo:

$$S_j = \sum_{i=1}^{n} S_{ij}.$$

5. Calcular el estadístico F_Q:

$$F_Q = \frac{(k-1) \sum_{j=1}^{k} S_j^2}{k \sum_{j=1}^{k} \sum_{i=1}^{n} (S_{ij} - \bar{S}_j)^2},$$

donde $\bar{S}_j$ es el promedio de S_j.

6. Comparar F_Q con el valor crítico de la distribución F con $k-1$ y $(k-1)(n-1)$ grados de libertad.

La prueba de Quade es útil cuando se sospecha que las diferencias entre los algoritmos varían de un problema a otro, y se desea dar más peso a los problemas donde esas diferencias son mayores.

4.2.2. Procedimientos post-hoc

Una vez que la prueba de Friedman (o una de sus variantes) indica que existen diferencias significativas entre los algoritmos, es útil identificar cuáles algoritmos difieren del método de control. Para ello, se emplean procedimientos *post-hoc* que ajustan los niveles de significancia para controlar el FWER.

El estadístico utilizado para comparar el algoritmo i con el algoritmo j es:

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6n}}},$$

donde R_i y R_j son los rangos promedio de los algoritmos i y j, respectivamente, obtenidos de la prueba de Friedman u otra prueba similar.

Los valores p asociados se obtienen de la distribución normal estándar $N(0,1)$ utilizando el valor absoluto de z. Sin embargo, estos valores p no son adecuados para comparaciones múltiples porque no tienen en cuenta las demás comparaciones en la familia de hipótesis. Por ello, es necesario ajustar los valores p para controlar el FWER.

A continuación, se describen varios procedimientos para ajustar los valores p. La notación utilizada es:

- Los índices i y j corresponden a comparaciones o hipótesis específicas dentro de la familia de hipótesis, ordenadas según sus valores p de forma ascendente. El índice i se refiere a la hipótesis cuya APV se está calculando, mientras que el índice j se refiere a otra hipótesis en la familia.

- p_j es el valor p obtenido para la hipótesis j.

Los procedimientos de ajuste de valores p se pueden clasificar en varias categorías:

4.2.2.1. Procedimientos de un solo paso

- **Bonferroni-Dunn**: Este método ajusta el nivel de significancia α en un solo paso dividiéndolo por el número de comparaciones realizadas $(k-1)$. Es el procedimiento más simple pero también el más conservador, lo que significa que tiene menos potencia estadística.

  El valor p ajustado (APV) para la hipótesis i se calcula como:

$$\text{APV}_i = \min\{(k-1)p_i, 1\}.$$

4.2.2.2. Procedimientos de paso descendente (step-down)

- **Holm**: Este procedimiento ajusta el nivel de significancia de manera secuencial descendente. Se ordenan los valores p de menor a mayor ($p_1 \leq p_2 \leq \cdots \leq p_{k-1}$), y se etiquetan las hipótesis correspondientes como $H_1, H_2, \ldots, H_{k-1}$.

  El método comienza comparando p_1 con $\alpha/(k-1)$. Si $p_1 \leq \alpha/(k-1)$, se rechaza H_1 y se procede a comparar p_2 con $\alpha/(k-2)$. Este proceso continúa hasta que se encuentra un p_i que no cumple la condición, momento en el cual se detiene y se acepta la hipótesis correspondiente y todas las restantes.

  El valor p ajustado para la hipótesis i se calcula como:

$$\text{APV}_i = \min \left\{ \max_{1 \leq j \leq i} [(k-j)p_j], 1 \right\}.$$

- **Holland**: Similar al método de Holm, pero utiliza una fórmula basada en la probabilidad acumulada. Rechaza las hipótesis H_1 a H_{i-1} si i es el menor entero tal que $p_i > 1 - (1-\alpha)^{k-i}$.

  El valor p ajustado se calcula como:

$$\text{APV}_i = \min \left\{ \max_{1 \leq j \leq i} [1 - (1-p_j)^{k-j}], 1 \right\}.$$

- **Finner**: También es un procedimiento de paso descendente que ajusta α utilizando una función exponencial.

  Rechaza las hipótesis H_1 a H_{i-1} si i es el menor entero tal que $p_i > 1 - (1-\alpha)^{(k-1)/i}$.

  El valor p ajustado se calcula como:

$$\text{APV}_i = \min \left\{ \max_{1 \leq j \leq i} \left[1 - (1-p_j)^{(k-1)/j}\right], 1 \right\}.$$

4.2.2.3. Procedimientos de paso ascendente (step-up)

- **Hochberg**: Este método ajusta el nivel de significancia de manera secuencial ascendente. Comienza comparando el valor p más grande p_{k-1} con α. Si $p_{k-1} \leq \alpha$, se rechaza la hipótesis correspondiente y todas las hipótesis con valores p menores. Si no, se compara p_{k-2} con $\alpha/2$, y así sucesivamente, hasta encontrar el primer p_i que cumple la condición.

  El valor p ajustado se calcula como:

$$\text{APV}_i = \max_{(k-1) \geq j \geq i} \left[(k-j)p_j\right].$$

- **Hommel**: Este procedimiento es más complejo y tiene mayor potencia estadística. Busca el mayor entero j para el cual $p_{k-j+i} > \alpha/j$ para todos $i = 1, \ldots, j$. Si no existe tal j, se rechazan todas las hipótesis; de lo contrario, se rechazan todas las hipótesis con $p_i \leq \alpha/j$.

  El cálculo del APV para la hipótesis i se realiza siguiendo un algoritmo específico que se puede encontrar en la literatura (debido a su complejidad, no se incluye aquí).

- **Rom**: Es una modificación del procedimiento de Hochberg diseñada para aumentar la potencia estadística. Funciona de la misma manera que el método de Hochberg, pero los valores de α se calculan mediante la siguiente expresión:

$$\alpha_{k-i} = \left[\sum_{j=1}^{i-1} \alpha_j - \sum_{j=1}^{i-2} \left(\frac{i}{k} \alpha_{i-j} \right) \right] / i,$$

donde $\alpha_{k-1} = \alpha$ y $\alpha_{k-2} = \alpha/2$. Los coeficientes r_{k-j} se obtienen de esta ecuación y se utilizan para calcular los APVs:

$$\text{APV}_i = \max_{(k-1) \geq j \geq i} [r_{k-j} p_j],$$

donde r_{k-j} se obtiene de la ecuación anterior (por ejemplo, $r = \{1, 2, 3, 3.814, 4.755, 5.705, 6.655, \dots\}$).

4.2.2.4. Procedimientos de dos pasos

- **Li**: Este procedimiento propone una estrategia de rechazo en dos pasos.
  1. **Paso 1**: Rechazar todas las hipótesis H_i si $p_{k-1} \leq \alpha$. De lo contrario, aceptar la hipótesis correspondiente a p_{k-1} y proceder al Paso 2.
  2. **Paso 2**: Rechazar cualquier H_i restante con $p_i \leq \left(\frac{1-p_{k-1}}{1-\alpha} \right) \alpha$.
  3. Los valores p ajustados se calculan como:

$$\text{APV}_i = \frac{p_i}{p_i + 1 - p_{k-1}}.$$

Estos procedimientos permiten ajustar los valores p obtenidos en las comparaciones múltiples, controlando el FWER y mejorando la validez estadística de los resultados. La elección del procedimiento adecuado depende del balance deseado entre el control del error tipo I y la potencia estadística. Los tests de comparación por parejas y múltiple proporcionan un marco robusto para realizar análisis estadísticos en los algoritmos que compararemos más adelante. Mediante el correcto uso de los tests, podremos diferenciar si los resultados obtenidos se deben a la aleatoriedad intrínseca del problema o si realemente un algoritmo es superior a otro.

Parte II.

Parte Informática

5. Componentes de los algoritmos implementados

En esta sección explicaremos con mayor profundidad la evolución diferencial, idea básica de los algoritmos que implementaremos más adelante, además explicaremos el funcionamiento de los algoritmos de agrupamiento diferencial y exploraremos dos métodos de búsqueda local que luego se utilizarán en el algoritmo SHADE-ILS

5.1. Evolución diferencial

Esta sección describe brevemente la Evolución Diferencial (DE). Similar a otros algoritmos evolutivos para la optimización numérica, una población de DE se representa como un conjunto de vectores de parámetros reales $\mathbf{x}_i = (x_1, \ldots, x_D), i = 1, \ldots, N$, donde D es la dimensionalidad del problema objetivo y N es el tamaño de la población.

Al comienzo de la búsqueda, los vectores individuales de la población se inicializan aleatoriamente. Luego, se repite un proceso de generación de vectores de prueba y selección hasta que se encuentra algún criterio de parada. En cada generación G, se genera un vector mutado $\mathbf{v}_{i,G}$ a partir de un miembro de la población existente $\mathbf{x}_{i,G}$ aplicando alguna estrategia de mutación. A continuación se muestran ejemplos de estrategias de mutación:

- **rand/1**
$$\mathbf{v}_{i,G} = \mathbf{x}_{r1,G} + F \cdot (\mathbf{x}_{r2,G} - \mathbf{x}_{r3,G})$$
- **rand/2**
$$\mathbf{v}_{i,G} = \mathbf{x}_{r1,G} + F \cdot (\mathbf{x}_{r2,G} - \mathbf{x}_{r3,G}) + F \cdot (\mathbf{x}_{r4,G} - \mathbf{x}_{r5,G})$$
- **best/1**
$$\mathbf{v}_{i,G} = \mathbf{x}_{best,G} + F \cdot (\mathbf{x}_{r1,G} - \mathbf{x}_{r2,G})$$
- **current-to-best/1**
$$\mathbf{v}_{i,G} = \mathbf{x}_{i,G} + F \cdot (\mathbf{x}_{best,G} - \mathbf{x}_{i,G}) + F \cdot (\mathbf{x}_{r1,G} - \mathbf{x}_{r2,G})$$

Los índices $r1, \ldots, r5$ se seleccionan aleatoriamente de $[1, N]$ de manera que difieran entre sí, así como i. $\mathbf{x}_{best,G}$ es el mejor individuo en la población en la generación G. El parámetro $F \in [0, 1]$ controla la potencia del operador de mutación diferencial, a mayor F, mayor será la diferencia entre el vector original y el mutado.

Después de generar el vector mutado $\mathbf{v}_{i,G}$, se cruza con el padre $\mathbf{x}_{i,G}$ para generar el vector de prueba $\mathbf{u}_{i,G}$. El cruce binomial, el operador de cruce más utilizado en DE, se implementa de la siguiente manera:

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{si rand}[0,1) \leq CR \text{ o } j = j_{\text{rand}} \\ x_{j,i,G} & \text{de lo contrario} \end{cases}$$

rand[0, 1) denota un número aleatorio seleccionado uniformemente de $[0, 1)$, y j_{rand} es un índice de variable de decisión que se selecciona aleatoriamente y de manera uniforme de $[1, D]$. $CR \in [0, 1]$ es la tasa de cruce.

Después de que se han generado todos los vectores de prueba $\mathbf{u}_{i,G}$, un proceso de selección determina los supervivientes para la siguiente generación. El operador de selección en DE estándar compara cada individuo $\mathbf{x}_{i,G}$ contra su vector de prueba correspondiente $\mathbf{u}_{i,G}$, manteniendo el mejor vector en la población.

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G} & \text{si } f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G}) \\ \mathbf{x}_{i,G} & \text{de lo contrario} \end{cases}$$

5.2. Búsqueda local

5.3. Algoritmos de descomposición

6. Algoritmos de comparación

Se explican los algoritmos que combinaremos para crear nuestra propuesta y que se utilizarán también para comparar como mejora el algoritmo final comparado con los algoritmos básicos. Se incluirá el pseudocódigo y la explicación de las partes esenciales que componen cada algoritmo.

6.1. SHADE

En esta sección explicaremos el algoritmo SHADE (Success-history based parameter adaptation for Differential Evolution), que es un componente clave del algoritmo SHADE-ILS que utilizaremos en nuestra propuesta. El artículo original donde se publicó este algoritmo puede encontrarse en [12].

SHADE (Success-History based Adaptive Differential Evolution) es una variante del algoritmo de Evolución Diferencial (DE) que utiliza un esquema de adaptación de parámetros basado en el historial de éxitos. A diferencia de otras variantes de DE, SHADE mantiene una memoria histórica de los valores de los parámetros de control que han sido exitosos en generaciones anteriores, y utiliza esta información para guiar la selección de los parámetros de control en generaciones futuras. El objetivo es mejorar la eficiencia de búsqueda y la capacidad de encontrar soluciones óptimas en problemas de optimización. Sus componentes principales que lo diferencian de la evolución estándar se describen a continuación.

Estrategia de mutación

La estrategia de mutación utilizada por SHADE es denominada current-to-p-best/1.

- **Estrategia current-to-pbest/1:**

$$v_{i,G} = x_{i,G} + F_i \cdot (x_{\text{pbest},G} - x_{i,G}) + F_i \cdot (x_{r1,G} - x_{r2,G})$$

El individuo $\mathbf{x}_{\text{pbest},G}$ es seleccionado del $N \cdot p$ ($p \in [0, 1]$) mejor de la generación G. F_i es el parámetro F usado por el individuo $\mathbf{x}_i$. Este parámetro p controla la voracidad del algoritmo, para balancear exploración con explotación. A menor p, mayor explotación.

En SHADE, cada individuo x_i tiene un p_i asociado, que se establece según la siguiente ecuación por generación:

$$p_i = \text{rand}[p_{\min}, 0.2]$$

donde $p_{\min}$ se establece de manera que cuando se selecciona el mejor individuo pbest, se seleccionen al menos 2 individuos, es decir, $p_{\min} = 2/N$. El valor máximo de 0.2 en la ecuación 6.1 es el valor máximo del rango para p sugerido.

Archivo Externo

Para mantener la diversidad, SHADE utiliza un archivo externo opcional. Los vectores padres $x_{i,G}$ que fueron peores que los vectores de prueba $u_{i,G}$ (y por lo tanto no son seleccionados para la supervivencia en el DE estándar) son preservados. Cuando se usa el archivo, $x_{r2,G}$ en la ecuación de mutación 6.1 es seleccionado de $P \cup A$, la unión de la población P y el archivo A. El tamaño del archivo se establece igual al de la población, es decir, $|A| = |P|$. Siempre que el tamaño del archivo excede $|A|$, los elementos seleccionados aleatoriamente son eliminados para hacer espacio para los nuevos elementos insertados.

Adaptación de Parámetros

SHADE utiliza un mecanismo de adaptación de parámetros basado en un registro histórico de configuraciones de parámetros exitosas. Para ello, SHADE mantiene una memoria histórica con H entradas para ambos parámetros de control de DE, CR y F, M_{CR} y M_{F}. Una representación de esta tabla se puede ver en 6.1. Al comienzo, el contenido de $M_{\mathrm{CR},i}$ y $M_{\mathrm{F},i}$ ($i = 1, ..., H$) se inicializa a 0.5.

En cada generación, los parámetros de control CR_i y F_i utilizados por cada individuo x_i se generan seleccionando primero un índice r_i aleatoriamente de $[1, H]$, y luego aplicando las siguientes ecuaciones:

$$CR_i = \mathrm{randn}(M_{\mathrm{CR},r_i}, 0.1)$$
$$F_i = \mathrm{randc}(M_{\mathrm{F},r_i}, 0.1)$$

Aquí, $\mathrm{randn}(\mu, \sigma^2)$ y $\mathrm{randc}(\mu, \sigma^2)$ son distribuciones normales y de Cauchy, respectivamente, con media μ y varianza σ^2.

Si los valores generados para CR_i están fuera del rango $[0, 1]$, se reemplazan por el valor límite (0 o 1) más cercano al valor generado. Cuando $F_i > 1$, F_i se trunca a 1, y cuando $F_i \leq 0$, se aplica repetidamente la ecuación 6.1 para intentar generar un valor válido.

En cada generación, los valores CR_i y F_i que logran generar un vector de prueba $u_{i,G}$ que es mejor que el individuo padre $x_{i,G}$ se registran como S_{CR} y S_F.

Los valores medios de S_{CR} y S_F para cada generación se almacenan en una memoria histórica M_{CR} y M_{F}. SHADE mantiene un conjunto diverso de parámetros para guiar la adaptación de parámetros de control a medida que avanza la búsqueda. Por lo tanto, incluso si S_{CR} y S_F para alguna generación particular contienen un conjunto deficiente de valores, los parámetros almacenados en la memoria de generaciones anteriores no pueden verse directamente afectados de manera negativa.

Al final de la generación, el contenido de la memoria se actualiza de la siguiente manera:

$$M_{\mathrm{CR},k,G+1} = \begin{cases} \mathrm{meanWA}(\mathrm{S}_{CR}) & \text{si } \mathrm{S}_{CR} \neq \emptyset \\ M_{\mathrm{CR},k,G} & \text{de lo contrario} \end{cases}$$

$$M_{\mathrm{F},k,G+1} = \begin{cases} \mathrm{meanWL}(\mathrm{S}_F) & \text{si } \mathrm{S}_F \neq \emptyset \\ M_{\mathrm{F},k,G} & \text{de lo contrario} \end{cases}$$

Un índice k ($1 \leq k \leq H$) determina la posición en la memoria a actualizar. Al comienzo de la búsqueda, k se inicializa a 1. k se incrementa cada vez que se inserta un nuevo elemento en el historial. Si $k > H$, k se establece en 1. En la generación G, se actualiza el k-ésimo elemento en la memoria. Nótese que cuando todos los individuos en la generación G no logran generar

un vector de prueba que sea mejor que el padre, es decir, $S_{CR} = S_F = \emptyset$, la memoria no se actualiza.
Además, la media ponderada meanWA(S_{CR}) y la media ponderada de Lehmer meanWL(S_F) se calculan usando las fórmulas descritas a continuación, y al igual que meanWA(S_{CR}), la cantidad de mejora se usa para influir en la adaptación de parámetros.

$$\text{meanWA}(S_{CR}) = \sum_{k=1}^{|S_{CR}|} w_k \cdot S_{CR,k}$$

$$w_k = \frac{\Delta f_k}{\sum_{k=1}^{|S_{CR}|} \Delta f_k}$$

donde $\Delta f_k = |f(u_{k,G}) - f(x_{k,G})|$.

$$\text{meanWL}(S_F) = \frac{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}}$$

Index	**1**	**2**	$\cdots$	**H - 1**	**H**
M_{CR}	$M_{CR,1}$	$M_{CR,2}$	$\cdots$	$M_{CR,H-1}$	$M_{CR,H}$
M_F	$M_{F,1}$	$M_{F,2}$	$\cdots$	$M_{F,H-1}$	$M_{F,H}$

Tabla 6.1.: La memoria histórica M_{CR}, M_F

Pseudocódigo de SHADE

Algorithm 6 SHADE

// Fase de inicialización
$G = 0$;
Inicializar población $P_0 = (x_{1,0}, \ldots, x_{N,0})$ aleatoriamente;
Establecer todos los valores en M_{CR}, M_F a 0.5;
Archivo $A = \emptyset$;
Contador de índice $k = 1$;
// Bucle principal
while No se cumplen los criterios de terminación **do**
  $S_{CR} = \emptyset$; $S_F = \emptyset$;
  for $i = 1$ to N **do**
    $r_i =$ Seleccionar aleatoriamente de $[1, H]$;
    $CR_{i,G} = \text{randn}_i(M_{CR,r_i}, 0.1)$;
    $F_{i,G} = \text{randc}_i(M_{F,r_i}, 0.1)$;
    $p_{i,G} = \text{rand}[p_{\min}, 0.2]$;
    Generar vector de prueba $u_{i,G}$ usando *current-to-pbest/1/bin*;
  end for
  for $i = 1$ to N **do**
    if $f(u_{i,G}) \leq f(x_{i,G})$ **then**
      $x_{i,G+1} = u_{i,G}$;
    else
      $x_{i,G+1} = x_{i,G}$;
    end if
    if $f(u_{i,G}) < f(x_{i,G})$ **then**
      $x_{i,G} \rightarrow A$;
      $CR_{i,G} \rightarrow S_{CR}$, $F_{i,G} \rightarrow S_F$;
    end if
  end for
  Si $|A| \geq |P|$, se eliminan individuos seleccionados aleatoriamente para que $|A| \leq |P|$;
  if $S_{CR} \neq \emptyset$ y $S_F \neq \emptyset$ **then**
    Actualizar $M_{CR,k}$, $M_{F,k}$ basado en S_{CR}, S_F;
    $k = k + 1$;
    if $k > H$ **then**
      k se establece en 1;
    end if
  end if
end while

6.2. SHADE-ILS

En este apartado, presentamos el algoritmo SHADE-ILS, expuesto por primera vez en [5], que combina el uso de la técnica de evolución diferencial SHADE explicada anteriormente y la búsqueda local. Además proporciona un método de reinicio para cuando se considera que la población se ha estancado en un óptimo local y no se puede mejorar más. Describiremos en detalle los elementos que componen el algoritmo y un pseudocódigo que nos proporcione una visión global del algoritmo.

Algorithm 7 SHADE-ILS

```
1: Algoritmo 1: SHADE-ILS
2: población ← random(dim, tamaño_población)
3: solución_inicial ← (superior + inferior)/2
4: actual_mejor ← BL(solución_inicial)
5: mejor_solución ← actual_mejor
6: while evaluaciones_totales < evaluaciones_máximas do
7:    previo ← actual_mejor.fitness
8:    actual_mejor ← SHADE(población, actual_mejor)
9:    mejora ← previo - actual_mejor.fitness
10:   Escoge el método de BL a aplicar en esta iteración.
11:   actual_mejor ← BL(población, actual_mejor)
12:   Actualiza probabilidad de aplicar BL.
13:   if mejor(actual_mejor, mejor_solución) then
14:      mejor_solución ← actual_mejor
15:   end if
16:   if Debe reiniciar then
17:      Reinicia y actualiza actual_mejor
18:   end if
19: end while
```

Método de exploración

Como su nombre indica, el algoritmo SHADE-ILS utiliza como método de exploración del espacio el algoritmo SHADE, explicado en detalle en la sección anterior.

Selección de la Búsqueda Local

La selección de la búsqueda local a utilizar en cada iteración se lleva a cabo según la mejora que ha aportado cada búsqueda local en su última aplicación. Inicialmente la mejora de cada método de búsqueda local es 0, así que en las primeras llamadas a la función de búsqueda local se aplique cada vez un método distinto hasta haber aplicado todos una vez. Cuando se ha aplicado un método cada vez, tenemos para cada método el ratio de mejora I_{LS}. A partir de ahora se aplicará siempre el método con mayor I_{LS} y se actualizará este valor en cada aplicación de la BL seleccionada. De esta forma se intentará aplicar siempre el método que mayor mejora aporta, cuando un método tenga un rendimiento peor, su I_{LS} disminuirá y otro método con mayor I_{LS} ocupará su lugar. Este método no garantiza aplicar siempre el método óptimo, pero proporciona una buena heurística para decidir que método aplicar y permite cambiar rápidamente de método de BL si otro método se estanca. El I_{LS} se calcula como:

$$\text{ILS} = \frac{\text{fitness(BeforeLS)} - \text{fitness(AfterLS)}}{\text{fitness(BeforeLS)}}$$

En [5] se propone utilizar dos métodos de búsqueda local: el algoritmo MTS LS-1 y L-BFGS-B. El primero está especialmente diseñado para problemas LSGO y es es apropiado para problemas separables pero es muy sensible a rotaciones, el segundo es menos potente, pero menos sensible a rotaciones.

Mecanismo de reinicio

El mecanismo de reinicio que se propone en [5] consiste en reiniciar la población cuando se da la condición de que durante tres iteraciones consecutivas, el ratio de mejora es menor del 5 %. En estos casos el mecanismo de reinicio aplicado sigue los siguientes pasos:

- Se selecciona aleatoriamente una solución sol.
- Se aplica una perturbación a sol que siga una distribución uniforme de media 0 y longitud del intervalo un 1 % del dominio de búsqueda:

  $$\text{currentbest} = \text{sol} + \text{rand}_i \cdot 0.01 \cdot (b - a)$$

  donde rand_i devuelve un número aleatorio $\text{rand}_i \in [-1, 1]$ y $[a, b]$ es el dominio de búsqueda.
- Los parámetros adaptativos de los métodos de BL se reinician a sus valores por defecto.

6.3. DG2

[7]

6.4. ERDG

7. Propuesta

7.1. ERDG-SHADE

7.2. ERDG-SHADE-ILS

8. Resultados

8.1. Resultados obtenidos

9. Conclusiones

9.1. Conclusiones extraídas del análisis de los datos

Bibliografía

[1] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2011.02.002. URL https://www.sciencedirect.com/science/article/pii/S2210650211000034.

[2] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi: 10.1109/ICNN.1995.488968.

[3] Antonio LaTorre, Daniel Molina, Eneko Osaba, Javier Poyatos, Javier Del Ser, and Francisco Herrera. A prescription of methodological guidelines for comparing bio-inspired optimization algorithms. *Swarm and Evolutionary Computation*, 67:100973, 2021. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2021.100973. URL https://www.sciencedirect.com/science/article/pii/S2210650221001358.

[4] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2019. doi: 10.1109/TEVC.2018.2868770.

[5] Daniel Molina, Antonio LaTorre, and Francisco Herrera. Shade with iterative local search for large-scale global optimization. In *Proceedings of the 2018 IEEE Congress on Evolutionary Computation*, pages 1252–1259, Rio de Janeiro, Brasil, July 2018.

[6] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, second edition, 2006. ISBN 978-0387303031.

[7] Mohammad Nabi Omidvar, Ming Yang, Yi Mei, Xiaodong Li, and Xin Yao. Dg2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 21(6):929–942, 2017. doi: 10.1109/TEVC.2017.2694221.

[8] Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. A review of population-based metaheuristics for large-scale black-box global optimization—part i. *IEEE Transactions on Evolutionary Computation*, 26(5):802–822, 2022. doi: 10.1109/TEVC.2021.3130838.

[9] Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. A review of population-based metaheuristics for large-scale black-box global optimization—part ii. *IEEE Transactions on Evolutionary Computation*, 26(5):823–843, 2022. doi: 10.1109/TEVC.2021.3130835.

[10] Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997. ISSN 1573-2916. doi: 10.1023/A:1008202821328. URL https://doi.org/10.1023/A:1008202821328.

[11] Yuan Sun, Michael Kirley, and Saman K. Halgamuge. A recursive decomposition method for large scale continuous optimization. *IEEE Transactions on Evolutionary Computation*, 22(5):647–661, 2018. doi: 10.1109/TEVC.2017.2778089.

[12] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78, 2013. doi: 10.1109/CEC.2013.6557555.

[13] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

[14] Ming Yang, Mohammad Nabi Omidvar, Changhe Li, Xiaodong Li, Zhihua Cai, Borhan Kazimipour, and Xin Yao. Efficient resource allocation in cooperative co-evolution for large-scale global optimization. *IEEE Transactions on Evolutionary Computation*, PP:1–1, December 2016. doi: 10.1109/TEVC.2016.2627581.

[15] Ming Yang, Aimin Zhou, Changhe Li, and Xin Yao. An efficient recursive differential grouping for large-scale continuous problems. *IEEE Transactions on Evolutionary Computation*, 25(1):159–171, 2021. doi: 10.1109/TEVC.2020.3009390.