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 2023-2024

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 2023-2024, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 15 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 impulsado la necesidad de resolver problemas cada vez más complejos, que dependen de un número creciente de variables. Esto hace necesario diseñar algoritmos capaces de encontrar soluciones para problemas de alta dimensionalidad. Un ejemplo de esto es la optimización de redes neuronales, donde es común tener que optimizar cientos e incluso millones 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. Este problema se conoce como *la maldición de la alta dimensionalidad*. Nos referimos a este fenómeno como la serie de dificultades que aparecen al aumentar el número de dimensiones (o variables) en un espacio, afectando negativamente el rendimiento y la eficacia de los algoritmos de optimización y análisis. A medida que crece la dimensionalidad, las distancias entre puntos pierden significado, el volumen del espacio aumenta exponencialmente, y los datos se dispersan, complicando así su análisis.

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, que requieren más tiempo y recursos para evaluar todas las combinaciones posibles de variables.

En este trabajo proponemos el estudio de una técnica 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 *SHADEils* 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. En la obtención y análisis de resultados, utilizaremos la plataforma web *Tacolab*, que permite realizar comparaciones sencillas entre algoritmos. Con los datos obtenidos, determinaremos en qué casos esta técnica puede resultar efectiva y en cuáles no.

Palabras clave: capacidad de cálculo, optimización, redes neuronales, alta dimensionalidad, maldición de la dimensionalidad, agrupamiento de variables, paralelismo, complejidad computacional, algoritmos metaheurísticos, Julia, benchmark, Tacolab.

Abstract

The increase in computing power has driven the need to solve increasingly complex problems that depend on a larger number of variables, making it necessary to design algorithms capable of finding solutions to high-dimensional problems. For example, the optimization of neural networks often involves optimizing hundreds or even millions of variables, which is either impossible or extremely costly using classical algorithms due to the exponential increase in difficulty associated with higher dimensionality. This problem is primarily due to *the curse of dimensionality*. This term refers to the difficulties that arise as the number of dimensions (or variables) in a space increases, negatively impacting the performance and effectiveness of optimization and analysis algorithms. As dimensionality grows, distances between points become less informative, the space's volume increases exponentially, and data becomes increasingly sparse, making analysis more challenging.

In high-dimensional problems, many techniques that work well in low-dimensional spaces become ineffective because the data occupies only a small fraction of the entire space. This impacts algorithms for search, classification, and optimization, which require more time and resources to process and evaluate all possible combinations of variables.

In this work, we propose studying a **variable grouping** technique that aims 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 can leverage parallelism and reduce computational complexity, as each subgroup can be optimized without directly affecting the others. This technique is particularly useful for problems where certain variables are highly correlated with each other but have little or no relation to other variables in the set.

We will study the use of this technique in algorithms specifically designed to solve high-dimensional problems, as well as in algorithms that work better in low dimensions. This will allow us to compare whether it is more efficient to focus on designing algorithms that perform increasingly well in high-dimensional spaces or to break the problem down into smaller subproblems.

To achieve this, we will introduce the necessary mathematical concepts for understanding how to solve minimization problems, as well as the theory behind variable grouping algorithms. We will also analyze which statistical tests we can use to obtain an objective and effective comparison and explain why they were chosen.

We will also present basic metaheuristic algorithms, which we will later combine to create our proposal. These include *SHADE* and *SHADEils* as optimization algorithms and *ERDG* as a variable grouping algorithm.

Additionally, we will develop a library in the *Julia* programming language that implements the algorithms used. This library aims not only to demonstrate whether the variable grouping technique is effective but also to contribute to the *Julia* community by providing a tool that other developers can use and improve upon in the future.

To compare the results, we will use the *CEC2013 LSGO* benchmark, specifically designed for large scale global optimization problems. For results analysis, we will use the *Tacolab* web

platform, which allows for straightforward comparisons between algorithms. Based on the data obtained, we will objectively determine in which cases this technique can be effective and in which it cannot.

Keywords: computing power, optimization, neural networks, high dimensionality, curse of dimensionality, variable grouping, parallelism, computational complexity, metaheuristic algorithms, Julia, benchmark, Tacolab.

Índice general

Agradecimientos III

Resumen V

Abstract VII

Introducción XIII

Presupesto XVII

I. Parte Matemática 1

1. Preliminares 3

2. Optimización numérica 5
2.1. Definiciones . . . 5
2.2. Teoremas . . . 5
2.3. Dificultades(Alta dimensionalidad) . . . 5
2.4. ¿Evolución Diferencial? . . . 5
2.5. Introducción a la Optimización Numérica . . . 5
2.5.1. Optimización Sin Restricciones . . . 5
2.5.2. Optimización Con Restricciones . . . 5
2.6. Métodos de Búsqueda Lineal . . . 6
2.6.1. Fundamentos Teóricos . . . 6
2.6.2. Criterios de Armijo y Wolfe . . . 6
2.7. Métodos de Newton y Quasi-Newton . . . 7
2.7.1. Método de Newton . . . 7
2.7.2. Métodos Quasi-Newton . . . 7
2.8. El Método L-BFGS-B . . . 8
2.8.1. Descripción del Método . . . 8
2.8.2. Algoritmo y Funcionamiento . . . 8
2.8.3. Ventajas y Limitaciones . . . 8
2.9. Optimización Global a Gran Escala . . . 9
2.9.1. Desafíos de la Optimización a Gran Escala . . . 9
2.9.2. Aplicaciones en Problemas de Gran Escala . . . 9
2.9.3. Consideraciones Especiales . . . 10
2.10. Conceptos Matemáticos Clave . . . 10
2.10.1. Gradiente y Hessiana . . . 10
2.10.2. Funciones Convexas . . . 10
2.11. Conclusiones . . . 10

3. Agrupamiento de variables **11**
3.1. Definiciones . . . 11
3.2. Teoremas . . . 12

4. Tests estadísticos **17**
4.1. Definición . . . 17
4.2. Tests . . . 17
4.2.1. Test 1 . . . 17
4.2.2. Test 2 . . . 17
4.2.3. Test 3 . . . 17
4.3. Relevancia en el contexto de este TFG . . . 17

II. Parte Informática **19**

5. Componentes de los algoritmos implementados **21**
5.1. Evolución diferencial . . . 21
5.2. Búsqueda local . . . 22
5.3. Algoritmos de descomposición . . . 22

6. Algoritmos de comparación **23**
6.1. SHADE . . . 23
6.2. SHADE-ILS . . . 27
6.3. DG2 . . . 28
6.4. ERDG . . . 28

7. Propuesta **29**
7.1. DG2-SHADE-ILS . . . 29
7.2. ERDG-SHADE-ILS . . . 29

8. Resultados **31**
8.1. Resultados obtenidos . . . 31

9. Conclusiones **33**
9.1. Conclusiones extraídas del análisis de los datos . . . 33

A. Ejemplo de apéndice **35**

Glosario **37**

Bibliografía **39**

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, debemos explorar otras soluciones.

En este trabajo, proponemos utilizar la técnica del **agrupamiento diferencial de variables**, que permite agrupar las variables en conjuntos, de forma que cada variable interactúa principalmente 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 algoritmos**: Implementación de algoritmos para optimización continua sin restricciones en el lenguaje de programación Julia.
- **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 SHADEils. Por último, unas secciones finales donde se exponen los resultados obtenidos y las conclusiones que hemos obtenido de los resultados.

Contexto 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*, *SHADEILS*, *ERDG* y *DG2*.

Introducción a la Optimización Global a Gran Escala

La *maldición de la dimensionalidad* ha sido un problema fundamental en la optimización global. Este problema se acentúa en la optimización global a gran escala, donde el número de variables y restricciones crece exponencialmente. El objetivo principal de las técnicas LSGO es mejorar la escalabilidad de los algoritmos de optimización a medida que el número de variables, n, y su dimensionalidad crece.

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.

Operadores de Muestreo y Variación

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) es un algoritmo popular en la optimización global debido a su simplicidad y efectividad. Variantes como *SHADE* y *SHADEILS* 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.
- **SHADEILS**: Extiende SHADE mediante la integración de estrategias de búsqueda local, mejorando la precisión en problemas de alta dimensionalidad.

Particle Swarm Optimization (PSO)

PSO 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*.

Modelado por Aproximación y Modelos Sustitutos

Dado que resolver directamente un problema de gran escala puede ser costoso, el modelado por aproximación crea un modelo simplificado de la función objetivo. Este enfoque es particularmente útil cuando se dispone de una función de alto coste computacional.

Métodos Explícitos para el Aprendizaje de Interacción de Variables

Los métodos explícitos para el aprendizaje de interacción buscan explotar las relaciones entre variables. Esto es fundamental en problemas donde la estructura del problema puede ser aprovechada.

ERDG y DG2

- **ERDG (Recursive Differential Grouping)**: ERDG se basa en la identificación de interacciones mediante particionamiento iterativo de variables. Esta técnica permite detectar grupos de variables interdependientes, lo cual facilita la resolución de subproblemas de forma eficiente.
- **DG2**: Es una técnica de descomposición que aprovecha las diferencias finitas para identificar interacciones. A diferencia de ERDG, DG2 ofrece una mayor precisión en la identificación de interacciones con una complejidad computacional más baja.

ERDG y DG2 son cruciales en el contexto de LSGO, ya que permiten dividir el problema en componentes independientes, lo cual facilita el proceso de optimización en problemas de gran escala.

Ventajas y Desventajas de las Técnicas

Según el teorema *No Free Lunch*, no existe un algoritmo universal que sea óptimo para todos los problemas. Por lo tanto, los enfoques híbridos que combinan varias estrategias tienden a ofrecer mejores resultados.

Hibridación en Algoritmos Evolutivos

La hibridación en algoritmos evolutivos permite:

1. Mejorar la velocidad de convergencia.
2. Incrementar la calidad de la solución.
3. Incorporar el algoritmo en sistemas más grandes.

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo `preliminares/introduccion.tex`

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

Derivada Direccional: Sea $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ una función diferenciable, y $\mathbf{u} = (u_1, \ldots, u_n)$ un vector de U_X. La derivada direccional de f en la dirección $\mathbf{u}$, denotada $D_{\mathbf{u}} f(\mathbf{x})$, se define como

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

Integral de Línea: Sea L una curva con puntos extremos A y B en el espacio de decisión $\mathbb{R}^n$ y la longitud del arco de L sea l. Sea C cualquier punto en L y la coordenada de $C(\mathbf{x})$ puede ser determinada de manera única por la longitud del arco $AC(s)$: $\mathbf{x} = \mathbf{x}(s)$, $s \in [0, l]$. La integral de una función $g : \mathbb{R}^n \to \overline{\mathbb{R}}$ a lo largo de la curva L se da por

$$\int_L g(\mathbf{x}) ds = \int_0^l g(\mathbf{x}(s)) ds.$$

2. Optimización numérica

2.1. Definiciones

2.2. Teoremas

2.3. Dificultades(Alta dimensionalidad)

2.4. ¿Evolución Diferencial?

2.5. Introducción a la 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 [**?**]. 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.

2.5.1. 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:

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

2.5.2. 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:

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

2.6. Métodos de Búsqueda Lineal

Los métodos de búsqueda lineal son técnicas iterativas utilizadas para encontrar un mínimo local de una función a lo largo de una dirección específica. Estos métodos son esenciales en algoritmos de optimización más complejos y son utilizados para determinar el tamaño de paso en cada iteración [**?**].

2.6.1. Fundamentos Teóricos

La idea principal es seleccionar una dirección de búsqueda $\mathbf{d}_k$ en el punto actual $\mathbf{x}_k$ y encontrar un escalar $\alpha_k > 0$ tal que:

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k).$$

Definición 2.2. Un **método de búsqueda lineal** busca resolver el siguiente problema unidimensional:

$$\alpha_k = \arg\min_{\alpha > 0} \phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{d}_k).$$

2.6.2. Criterios de Armijo y Wolfe

Para garantizar la convergencia y evitar pasos demasiado largos o cortos, se emplean condiciones de aceptabilidad para el tamaño de paso α_k [**?**].

Definición 2.3. La **condición de Armijo** establece que α_k debe satisfacer:

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

donde $0 < c_1 < 1$ es un parámetro pequeño, típicamente $c_1 = 10^{-4}$.

Definición 2.4. Las **condiciones de Wolfe** son dos y deben cumplirse simultáneamente:

1. **Condición de Armijo** (como antes).
2. **Condición de curvatura**:

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

donde $c_1 < c_2 < 1$.

Estas condiciones garantizan que el tamaño de paso α_k no sea demasiado pequeño ni demasiado grande, promoviendo una convergencia eficiente.

2.7. Métodos de Newton y Quasi-Newton

Los métodos de Newton y quasi-Newton son algoritmos iterativos utilizados para encontrar los mínimos (o máximos) de funciones diferenciables. Se basan en aproximaciones de la función y su curvatura [**?**].

2.7.1. Método de Newton

El método de Newton utiliza información de segunda derivada (la matriz Hessiana) para encontrar la dirección de descenso óptima.

Definición 2.5. En cada iteración k, el **método de Newton** actualiza la solución mediante:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k),$$

donde $\nabla^2 f(\mathbf{x}_k)$ es la matriz Hessiana de f en $\mathbf{x}_k$.

2.7.1.1. Ventajas y Limitaciones

El método de Newton tiene una convergencia cuadrática cuando está cerca del mínimo, pero requiere calcular y almacenar la matriz Hessiana, lo cual es computacionalmente costoso para problemas de gran escala.

2.7.2. Métodos Quasi-Newton

Para superar las limitaciones del método de Newton, los métodos quasi-Newton construyen una aproximación de la matriz Hessiana utilizando información de las derivadas de primer orden.

Definición 2.6. Los **métodos quasi-Newton** generan una secuencia de matrices $\mathbf{B}_k$ que aproximan la Hessiana, actualizando la solución como:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k).$$

2.7.2.1. Actualizaciones de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Una de las fórmulas de actualización más utilizadas es la de BFGS [**?**].

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

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

2.7.2.2. Ventajas y Limitaciones

Los métodos quasi-Newton tienen una convergencia superlineal y evitan el cálculo de la Hessiana exacta. Sin embargo, aún requieren almacenar matrices de tamaño $n \times n$, lo cual puede ser problemático para n grande.

2.8. El Método L-BFGS-B

El método L-BFGS-B (*Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Bounds*) es una variante del algoritmo BFGS de memoria limitada que incorpora límites en las variables [**?**].

2.8.1. Descripción del Método

L-BFGS-B es un método de optimización quasi-Newton de memoria limitada que maneja restricciones de caja en las variables.

Definición 2.7. El método L-BFGS-B resuelve problemas de la forma:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad \text{sujeto a} \quad l_i \leq x_i \leq u_i, \quad i = 1, \ldots, n,$$

donde l_i y u_i son los límites inferiores y superiores, respectivamente.

2.8.2. Algoritmo y Funcionamiento

El método L-BFGS-B utiliza una aproximación de la matriz Hessiana basada en un número limitado de vectores $\mathbf{s}_k$ y $\mathbf{y}_k$, evitando el almacenamiento completo de matrices.

Algorithm 1 Algoritmo L-BFGS-B

1: Inicializar $\mathbf{x}_0$, límites l_i, u_i, memoria m
2: **for** $k = 0, 1, 2, \ldots$ hasta convergencia **do**
3: Calcular el gradiente $\nabla f(\mathbf{x}_k)$
4: Aplicar proyección para manejar las restricciones de caja
5: Calcular la dirección de búsqueda $\mathbf{d}_k$ usando la aproximación de la Hessiana
6: Realizar una búsqueda lineal para encontrar α_k
7: Actualizar $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$
8: Actualizar los vectores de memoria $\mathbf{s}_k$ y $\mathbf{y}_k$
9: **end for**

2.8.3. Ventajas y Limitaciones

Las principales ventajas de L-BFGS-B incluyen:

- **Eficiencia en problemas de gran escala**: Al utilizar memoria limitada, es adecuado para problemas con gran número de variables.
- **Manejo de restricciones simples**: Las restricciones de caja son manejadas eficientemente mediante proyecciones.

Limitaciones:

- **No adecuado para restricciones complejas**: El método está diseñado para restricciones de caja y puede no ser eficiente con restricciones más generales.
- **Sensibilidad a la elección de parámetros**: La performance puede depender de la elección de parámetros como el tamaño de memoria m.

Teorema 2.8. *Si la función objetivo f es convexa y dos veces continuamente diferenciable, y si las matrices aproximadas $\mathbf{B}_k$ son positivas definidas, entonces el método L-BFGS-B converge globalmente al mínimo dentro de las restricciones dadas [**? **].*

2.9. Optimización Global a Gran Escala

La optimización global a gran escala se refiere a la resolución de problemas con un gran número de variables y múltiples óptimos locales. Estos problemas son comunes en áreas como el aprendizaje automático, donde los modelos pueden tener millones de parámetros [**?**].

2.9.1. Desafíos de la Optimización a Gran Escala

Los principales desafíos incluyen:

- **Dimensionalidad alta**: El almacenamiento y el cálculo de matrices pueden ser computacionalmente costosos.
- **Múltiples óptimos locales**: Es difícil garantizar que se alcance el óptimo global.
- **Restricciones complejas**: Manejar restricciones en un espacio de alta dimensión puede ser complicado.

2.9.2. Aplicaciones en Problemas de Gran Escala

El método L-BFGS-B es especialmente útil en problemas de gran escala debido a:

- **Memoria limitada**: Utiliza una cantidad fija de memoria, independientemente del número de variables.
- **Eficiencia computacional**: Reduce el costo computacional al evitar operaciones costosas.

Definición 2.9. Una función es **parcialmente aditivamente separable** si es de la siguiente forma:

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

donde $f_i(\cdot)$ son subfunciones que solo dependen de las variables que forman cada vector $\mathbf{x}_i$.

Teorema 2.10. *Sea $f(\mathbf{x})$ una función (parcialmente) aditivamente separable. Para todo $a, b_1 \neq b_2, \delta \in \mathbb{R}, \delta \neq 0$, si se cumple la siguiente condición:*

$$\Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](\mathbf{x})|_{x_p=a,x_q=b_2} \tag{2.1}$$

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.

2.9.3. Consideraciones Especiales

En problemas de gran escala, es crucial aprovechar la estructura de la función objetivo, como la separabilidad, para mejorar la eficiencia de los algoritmos [**?**].

2.10. Conceptos Matemáticos Clave

Para comprender completamente los métodos de optimización discutidos, es importante familiarizarse con varios conceptos matemáticos.

2.10.1. Gradiente y Hessiana

Definición 2.11. El **gradiente** de una función $f : \mathbb{R}^n \to \mathbb{R}$ es el vector de derivadas parciales:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^\top.$$

Definición 2.12. La **matriz Hessiana** es la matriz de segundas derivadas parciales:

$$\nabla^2 f(\mathbf{x}) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{i,j=1}^{n}.$$

El gradiente indica la dirección de mayor incremento de la función, mientras que la Hessiana proporciona información sobre la curvatura de la función.

2.10.2. Funciones Convexas

Definición 2.13. Una función $f : \mathbb{R}^n \to \mathbb{R}$ es **convexa** si para todo $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ y $\lambda \in [0, 1]$ se cumple:

$$f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

La convexidad es una propiedad deseable en optimización, ya que garantiza que cualquier mínimo local es también mínimo global.

2.11. Conclusiones

La optimización numérica es un campo amplio y esencial en muchas áreas de la ciencia y la ingeniería. Los métodos de Newton y quasi-Newton proporcionan herramientas poderosas para encontrar óptimos locales de funciones diferenciables. El método L-BFGS-B, en particular, es altamente efectivo para problemas de gran escala con restricciones simples, gracias a su uso eficiente de la memoria y su capacidad para manejar límites en las variables.

La comprensión de conceptos matemáticos como el gradiente, la Hessiana y la convexidad es fundamental para aplicar estos métodos correctamente. Además, aprovechar la estructura de la función objetivo, como la separabilidad, puede mejorar significativamente la eficiencia de los algoritmos de optimización.

En resumen, los métodos y conceptos presentados son fundamentales para abordar problemas de optimización global a gran escala y continúan siendo un área activa de investigación y aplicación.

3. Agrupamiento de variables

Inspirados por la metodología del análisis clúster, estudiaremos los fundamentos teóricos del agrupamiento diferencial, una técnica de agrupamiento 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 clústers de forma que en cada clúster los datos sean lo más parecidos posibles y distintos del resto de clúster, el objetivo de esta técnica es separar las variables en conjuntos, de forma que cada conjunto 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 de variables lo que agrupamos son las variables acorde a la dependencia que existe entre ellas.

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

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

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

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

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

□

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

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. □

4. Tests estadísticos

4.1. Definición

4.2. Tests

4.2.1. Test 1

4.2.2. Test 2

4.2.3. Test 3

4.3. Relevancia en el contexto de este TFG

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 [TF13].

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 2 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 [MLH18], 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 3 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 [MLH18] 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 [MLH18] 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

[OYM+17]

6.4. ERDG

7. Propuesta

7.1. DG2-SHADE-ILS

7.2. ERDG-SHADE-ILS

8. Resultados

8.1. Resultados obtenidos

9. Conclusiones

9.1. Conclusiones extraídas del análisis de los datos

A. Ejemplo de apéndice

Los apéndices son opcionales.

Este fichero `apendice-ejemplo.tex` es una plantilla para añadir apéndices al TFG. Para ello, es necesario:

- Crear una copia de este fichero `apendice-ejemplo.tex` en la carpeta `apendices` con un nombre apropiado (p.e. `apendice01.tex`).
- Añadir el comando `\input{apendices/apendice01}` en el fichero principal `tfg.tex` donde queremos que aparezca dicho apéndice (debe de ser después del comando `\appendix`).

Glosario

La inclusión de un glosario es opcional.
Archivo: `glosario.tex`

$\mathbb{R}$ Conjunto de números reales.

$\mathbb{C}$ Conjunto de números complejos.

$\mathbb{Z}$ Conjunto de números enteros.

Bibliografía

[MLH18] Daniel Molina, Antonio LaTorre, y Francisco Herrera. Shade with iterative local search for large-scale global optimization. En *Proceedings of the 2018 IEEE Congress on Evolutionary Computation*, páginas 1252–1259, Rio de Janeiro, Brasil, July 2018.

[OYM+17] Mohammad Nabi Omidvar, Ming Yang, Yi Mei, Xiaodong Li, y 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.

[TF13] Ryoji Tanabe y Alex Fukunaga. Success-history based parameter adaptation for differential evolution. En *2013 IEEE Congress on Evolutionary Computation*, páginas 71–78, 2013.