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 1 de julio de 2024

Fdo: Jesús García León

Agradecimientos

Agradecimientos (opcional, ver archivo `preliminares/agradecimiento.tex`).

Índice general

Resumen

En este trabajo se pretende estudiar las propiedades teóricas de los algoritmos utilizados para resolver problemas de optimización, poniendo un énfasis en los algoritmos utilizados para la optimización en alta dimensión, debido al crecimiento exponencial en complejidad que el aumento de dimensión suele acarrear. También se analizarán distintos test estadísticos y sus propiedades teóricas . Además, se pretende iniciar una biblioteca de algoritmos para dicho tipo de problemas. Se utilizarán técnicas metaheurísticas como algoritmos evolutivos basados en el algoritmo SHADE y se utilizarán técnicas de agrupamiento de variables que permitan descomponer el problema en subproblemas independientes. Combinando estas técnicas, se pretende desarrollar un algoritmo que supere a los anteriores. Finalmente se comparará el algoritmo resultante con sus versiones básicas para comprobar si efectivamente se obtienen mejores resultados.

File: `preliminares/resumen.tex`

Abstract

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.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`

Introducción

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`

Parte I.

Parte Matemática

1. Problemas de Optimización

1.1. Definiciones

1.2. Algoritmos

1.3. Dificultades

2. 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.

2.1. Definiciones

2.2. Teoremas

2.3. Problemas descomponibles mediante agrupamiento de variables

3. Algoritmos de optimización en alta dimensión

3.1. Definición

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. Metaheurísticas

5.1. Definición

Explicar las metaheurísticas en general, diferencias con las heurísticas y su clasificación. Nos centraremos en los algoritmos evolutivos, en los de descomposición y de búsqueda local

5.2. Algoritmos evolutivos

5.2.1. 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.

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

5.3. Algoritmos de descomposición

5.4. Búsqueda local

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 (o 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 1 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 2 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. Una vez 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 esta 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. RDG2

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.