Manual de usuario
SOLVER BIO-INSPIRADOS

José Lara Arce
Pontificia Universidad Católica de Valparaíso, Chile
jose.lara.a01@mail.pucv.cl

March 16, 2025

Índice

1. Dependencias **2**
1.1. Activar el entorno virtual 2
1.2. Configurar Visual Studio Code 2
1.3. Instalar dependencias 3

2. Introducción al solver **3**

3. Estructura Interna del Solver **4**
3.1. Archivos Principales 5

4. Ejemplo de uso y explicación **6**
4.1. Optimización de funciones matemáticas con el algoritmo SBOA 6
4.1.1. Paso 1: Configuración del Archivo `experiments_config.json` 6
4.1.2. Paso 2: Poblar la base de datos 8
4.1.3. Paso 3: Ejecutar el solver 9

1. Dependencias

Para el correcto funcionamiento del solver, es necesario contar con los siguientes requisitos:

- **Versión de Python**: 3.11.9 (64 bits)
- **IDEs recomendados**: Visual Studio Code, PyCharm
- **Gestión de dependencias**: Se utiliza la librería `virtualenv`, que puede instalarse con:

```
pip install virtualenv
```

Nota: La instalación de un entorno virtual es opcional. Si no desea aislar las dependencias del proyecto y prefiere instalar los paquetes de Python directamente en el entorno global del sistema, puede omitir este paso y proceder directamente a la sección 1.3. Sin embargo, se recomienda el uso de un entorno virtual para evitar conflictos entre versiones de librerías en diferentes proyectos.

1.1. Activar el entorno virtual

`virtualenv` permite crear entornos virtuales aislados para gestionar paquetes sin afectar otras configuraciones del sistema. Para ello:

1. Cree un entorno virtual con el siguiente comando:

```
virtualenv -p python3 env
```

2. Una vez creado, active el entorno virtual ejecutando:

```
env\Scripts\activate
```

Al activarse correctamente, la consola debería mostrar algo similar a:

```
(env) C:\Users\Usuario\Proyecto>
```

1.2. Configurar Visual Studio Code

Para asegurarse de que Visual Studio Code use el intérprete correcto:

1. Abra la **Command Palette** con:

```
Ctrl + Shift + P
```

2. Escriba `>Select Interpreter` y seleccione la opción que contenga `env`.

1.3. Instalar dependencias

Para instalar las dependencias del proyecto, asegúrese de que el entorno virtual esté activado y ejecute en consola:

```
pip install -r requirements.txt
```

2. Introducción al solver

El solver que presentamos en este manual está diseñado para resolver problemas de optimización utilizando técnicas de metaheurísticas. Estas técnicas son métodos de búsqueda de soluciones aproximadas para problemas donde las soluciones exactas son difíciles o imposibles de obtener debido a la complejidad computacional. En particular, el solver implementa las siguientes metaheurísticas hasta la fecha:

- Arithmetic Optimization Algorithm (AOA)
- Enhanced Beluga Whale Optimization Algorithm (EBWOA)
- Elk Herd Optimizer (EHO)
- Eurasian Oystercatcher Optimizer (EOO)
- Flilled Lizard Optimization (FLO)
- Fox-inspired Optimization Algorithm (FOX)
- Genetic Algorithm (GA)
- Gannet Optimization Algorithm (GOA)
- Grey Wolf Optimizer (GWO)
- Fox-inspired Optimization Algorithm (FOX)
- Genetic Algorithm (GA)
- Honey Badger Algorithm (HBA)
- Horned Lizard Optimization Algorithm (HLOA)
- Lyrebird Optimization Algorithm (LOA)
- Narwhal Optimization (NO)
- Puma Optimizer (PO)

- Pendulum Search Algorithm (PSA)
- Particle Swarm Optimization (PSO)
- Quokka Optimization Algorithm (QSO)
- Reptile Search Algorithm (RSA)
- Secretary Bird Optimization Algorithm (SBOA)
- Sine Cosine Algorithm (SCA)
- Sea-Horse Optimizer (SHO)
- Tasmanian Devil Optimization (TDO)
- Whale Optimization Algorithm (WOA)
- Wombat Optimization Algorithm (WOM)

El solver permite al usuario optimizar diferentes funciones objetivo, que se definen en el archivo de configuración.

3. Estructura Interna del Solver

El solver está organizado en módulos con el objetivo de facilitar su mantenimiento y expansión. A continuación, se describen los componentes clave de la estructura del código:

- **BD**: Módulo encargado de gestionar la base de datos, así como su correspondiente constructor.
- **Discretization**: Contiene los elementos responsables de la binarización de las soluciones.
- **Diversity**: Incluye las métricas utilizadas para evaluar y visualizar la diversidad de las soluciones generadas por el solver.
- **Explicaciones_Manuales**: Reúne archivos PDF con explicaciones detalladas sobre el funcionamiento del solver y los problemas que se abordan a lo largo de la asignatura.
- **Graficos_Benchmark**: Contiene archivos PDF con gráficos que ilustran las funciones matemáticas, permitiendo a los usuarios visualizar la complejidad de las funciones que están optimizando.
- **Metaheuristics**: Incluye los archivos Python (`.py`) correspondientes a las diversas metaheurísticas implementadas en el solver.

- **Problem**: Contiene las instancias de los problemas abordados en la asignatura, como BEN, SCP y USCP.
- **Solver**: Agrupa los códigos que implementan los solvers específicos para cada uno de los problemas mencionados.
- **Util**: Contiene archivos de utilidad empleados por los módulos anteriores para garantizar su correcto funcionamiento.
- **Util/json**: Incluye archivos de configuración necesarios para realizar experimentos, así como archivos relacionados con las rutas y parámetros correspondientes.

3.1. Archivos Principales

Los siguientes son los archivos más importantes dentro del proyecto, cada uno con una función específica:

- **analisisBEN.py (analisisSCP, analisisUSCP)**: Archivos Python ejecutables utilizados para realizar análisis sobre ejecuciones previas. Estos archivos son útiles para presentaciones o para verificar que la metaheurística esté funcionando correctamente.
- **crearBD.py**: Crea la base de datos si esta no ha sido creada previamente.
- **levantarCMD.py**: Permite la creación de múltiples instancias de la línea de comandos (CMD) según las necesidades del usuario, facilitando la ejecución paralela del solver.
- **limpiarEntorno.py**: Elimina todos los archivos dentro de la carpeta de resultados, garantizando un entorno limpio para nuevas ejecuciones.
- **main.py**: El programa principal que coordina y ejecuta las funcionalidades del solver.
- **poblarDB.py**: Archivo encargado de poblar la base de datos con los experimentos previamente configurados.
- **reiniciarDB.py**: Permite reiniciar la base de datos, eliminando los experimentos existentes y configurando las instancias por defecto.
- **reiniciarSolver.py**: Reinicia el solver, restaurándolo a su estado inicial, similar a cuando se descarga por primera vez. Este proceso incluye la ejecución de `limpiarEntorno.py` y `reiniciarDB.py`. Será de los archivos con más utilidad para ustedes.

4. Ejemplo de uso y explicación

A continuación, se presenta un ejemplo detallado de cómo utilizar el solver para la optimización de funciones matemáticas mediante el algoritmo SBOA.

4.1. Optimización de funciones matemáticas con el algoritmo SBOA

Para comenzar con la optimización, lo primero que deben hacer es seleccionar las funciones que desean optimizar. En este ejemplo, vamos a optimizar las funciones 'F1', 'F8', 'F9' y 'F16'.

4.1.1. Paso 1: Configuración del Archivo `experiments_config.json`

Diríjanse al directorio `'Util/json'` y abran el archivo `experiments_config.json`. Dentro de este archivo, encontrarán varias configuraciones que deberán ajustar de acuerdo con los problemas y parámetros específicos. A continuación, se detallan los cambios necesarios para este caso:

- Configuración de funciones BEN

  Para trabajar con funciones matemáticas, deben asegurarse de que la opción `'ben'` esté configurada en `true`, ya que esto indica que se optimizarán funciones BEN (Benchmark):

  ```
  "ben": true,
  "scp": false,
  "uscp": false,
  ```

- Selección de metaheurísticas

  A continuación, deben especificar la metaheurística que desean utilizar. En este caso, seleccionamos `SBOA` (Secretary Bird Optimization Algorithm). Este parámetro puede contener una lista de varias metaheurísticas:

  ```
  "mhs": ["SBOA"],
  ```

- Configuración de acciones de discretización (para problemas binarios)

  La sección `DS_actions` es relevante solo para problemas binarios, como SCP y USCP. Dado que estamos trabajando con BEN, pueden dejar este parámetro tal cual está, ya que no afecta la optimización de funciones matemáticas:

  ```
  "DS_actions": ["S3-ELIT", "V3-ELIT"],
  ```

- Especificación de dimensiones

  En la sección `dimensiones`, se especifican las dimensiones de las instancias de cada función matemática. A continuación, se detallan las dimensiones de las funciones BEN que se optimizarán. Si desean, pueden modificar la dimensión, pero tengan en cuenta que a mayor dimensión, mayor será el tiempo de cómputo y la calidad de las soluciones podría no ser óptima. En nuestro caso, optimizaremos las funciones 'F1', 'F8', 'F9' y 'F16':

```
"dimensiones": {
    "BEN": {
        "F1": [30], "F2": [30], "F3": [30],
        "F4": [30], "F5": [30], "F6": [30],
        "F7": [30], "F8": [30], "F9": [30],
        "F10": [30], "F11": [30], "F12": [30],
        "F13": [30], "F14": [2], "F15": [4],
        "F16": [2], "F17": [2], "F18": [2],
        "F19": [3], "F20": [6], "F21": [4],
        "F22": [4], "F23": [4]
    }
},
```

- Selección de dimensiones para las funciones específicas

  En esta sección, se debe especificar la dimensión de las funciones que se van a optimizar. En el caso de las funciones seleccionadas, "F1", "F8", "F9z "F16", hemos optado por la dimensión 30 para cada una de ellas, excepto por "F16", que tiene dimensión 2. Esto asegura que el proceso de optimización tenga un espacio de búsqueda adecuado para encontrar la solución óptima.

- Consideraciones sobre la dimensión de las funciones

  Es importante tener en cuenta que aumentar la dimensión de las funciones puede incrementar el tiempo de cómputo, por lo que es recomendable elegir dimensiones que equilibren la eficiencia computacional y la precisión de la optimización.

- Configuración de parámetros de experimentos

  En la sección `experimentos`, se definen los parámetros específicos de los experimentos, tales como el número de iteraciones, la población y el número de experimentos a ejecutar. Para este ejemplo, trabajaremos con 100 iteraciones, una población de 50 y un único experimento:

```
"experimentos": {
    "BEN": {"iteraciones": 100, "poblacion": 50,
    "num_experimentos": 1},

    "SCP": {"iteraciones": 100, "poblacion": 10,
    "num_experimentos": 31},

    "USCP": {"iteraciones": 100, "poblacion": 10,
    "num_experimentos": 31}
},
```

- Número de iteraciones

  El parámetro de `iteraciones` indica cuántas veces se ejecutará el algoritmo de optimización. En este caso, hemos establecido 100 iteraciones, lo que debería ser suficiente para encontrar una buena aproximación de la solución óptima.

- Tamaño de la población

  El parámetro `poblacion` especifica el número de soluciones (individuos) en cada iteración. Hemos configurado una población de 50, lo cual es adecuado para que el algoritmo explore el espacio de soluciones de manera efectiva sin ser demasiado costoso computacionalmente.

- Número de experimentos

  El parámetro `num_experimentos` indica cuántas veces se debe ejecutar el experimento para obtener resultados estadísticamente significativos. En este caso, configuramos 1 experimento, ya que estamos interesados en ver cómo el algoritmo se comporta en una sola ejecución.

- Selección de instancias a optimizar

  Finalmente, en la sección `instancias`, deben seleccionar las instancias que desean optimizar. En este caso, seleccionaremos las funciones 'F1', 'F8', 'F9' y 'F16' del conjunto BEN:

```
"instancias": {
    "BEN": ["F1", "F8", "F9", "F16"],
    "SCP": ["41", "51", "61"],
    "USCP": ["uclr10", "uclr11"]
},
```

4.1.2. Paso 2: Poblar la base de datos

Una vez que hayan configurado el archivo `experiments_config.json`, el siguiente paso es poblar la base de datos. Para ello, deben dirigirse al archivo

`poblarDB.py` y ejecutarlo. Este script se encarga de preparar las instancias de los problemas y las configuraciones necesarias para el experimento.

4.1.3. Paso 3: Ejecutar el solver

Finalmente, deben ir al archivo principal `main.py` y ejecutarlo. Al hacerlo, podrán observar cómo el solver optimiza las soluciones en tiempo real a través de la consola.

Durante la ejecución, se mostrará el progreso de las optimizaciones, y podrán ver cómo las soluciones evolucionan conforme avanzan las iteraciones.