

Manual de usuario para ejecuciones de Set Covering Problem con AOA y GWO y su posterior análisis de resultados – Extensión de Solver Set Covering Problem

José Urbina Moreno

Pontificia Universidad Católica de Valparaíso

El presente manual hace referencia al código público disponible en el repositorio GitHub alojado en <https://github.com/Joseummmm/trabajoTitulo.git> el cual es una directa extensión del Solver desarrollado por

Felipe Cisternas Caneo, alojado en el siguiente repositorio público:

https://github.com/FelipeCisternasCaneo/Solver_SCP.git

1. Resumen

El presente manual explicará el flujo de trabajo necesario para llevar a cabo la ejecución de una metaheurística para la resolución del *Set Covering Problem* (SCP) (Karp, 1972) utilizando el *Arithmetic Optimization Algorithm* (AOA) (Abualigah, Diabat, Mirjalilid, Abd Elazizf, & Gandomih, 2021) y el *Grey Wolf Optimizer* (GWO) (Mirjalili, Mirjalili, & Lewis, 2014), para después analizar los resultados obtenidos y obtener diversas métricas y gráficos a partir de las mejores iteraciones de cada experimento.

2. Prerrequisitos

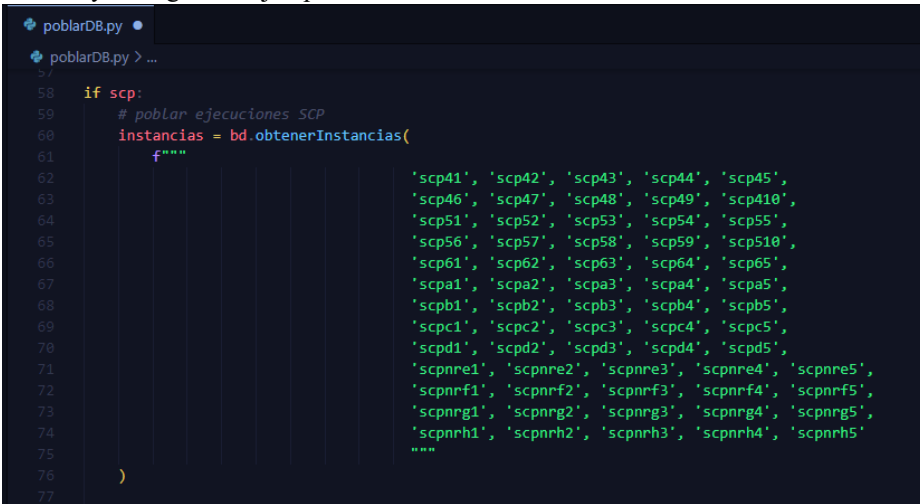
Para lograr la ejecución de los archivos de Python presentes en el solver, necesitaremos instalar los siguientes elementos:

- Python de 64 bits: Se debe instalar a través de [Microsoft Store](#), la [web oficial de Python](#) o a través de su gestor de paquetes de preferencia.
- Las siguientes librerías de Python, posterior a la instalación de Python y reinicio de su equipo. Por cada librería, se incluirá el comando a ejecutar en su terminal:
 - *Numpy*: **pip install numpy**
 - *Scipy*: **pip install scipy**
 - *Matplotlib*: **pip install matplotlib**
 - *Pandas*: **pip install pandas**
 - *Seaborn*: **pip install seaborn**

3. Flujo trabajo para la ejecución del problema

1. Poblar base de datos con las instancias a resolver:
 - a. Abrir archivo poblarDB.py
 - b. Buscar variable “instancias” dentro del if scp.
 - c. Poblar la lista que se le pasará a la función bd.obtenerInstancias() con las instancias que queremos resolver. Se incluye el siguiente ejemplo como referencia:

i.

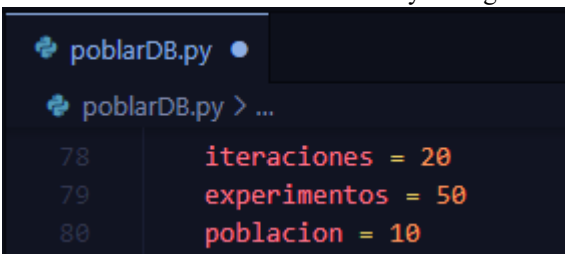


```
58 if scp:
59     # poblar ejecuciones SCP
60     instancias = bd.obtenerInstancias(
61         f"""
62         'scp41', 'scp42', 'scp43', 'scp44', 'scp45',
63         'scp46', 'scp47', 'scp48', 'scp49', 'scp410',
64         'scp51', 'scp52', 'scp53', 'scp54', 'scp55',
65         'scp56', 'scp57', 'scp58', 'scp59', 'scp510',
66         'scp61', 'scp62', 'scp63', 'scp64', 'scp65',
67         'scpa1', 'scpa2', 'scpa3', 'scpa4', 'scpa5',
68         'scpb1', 'scpb2', 'scpb3', 'scpb4', 'scpb5',
69         'scpc1', 'scpc2', 'scpc3', 'scpc4', 'scpc5',
70         'scpd1', 'scpd2', 'scpd3', 'scpd4', 'scpd5',
71         'scpnre1', 'scpnre2', 'scpnre3', 'scpnre4', 'scpnre5',
72         'scpnrf1', 'scpnrf2', 'scpnrf3', 'scpnrf4', 'scpnrf5',
73         'scpnrg1', 'scpnrg2', 'scpnrg3', 'scpnrg4', 'scpnrg5',
74         'scpnrh1', 'scpnrh2', 'scpnrh3', 'scpnrh4', 'scpnrh5'
75         """
76     )
77
```

Ilustración 1. Ejemplo de ingreso de instancias a resolver.

- d. Indicar las cantidad de iteraciones por experimento, cantidad de experimentos y tamaño de población a utilizar al resolver el SCP. Se incluye el siguiente ejemplo como referencia:

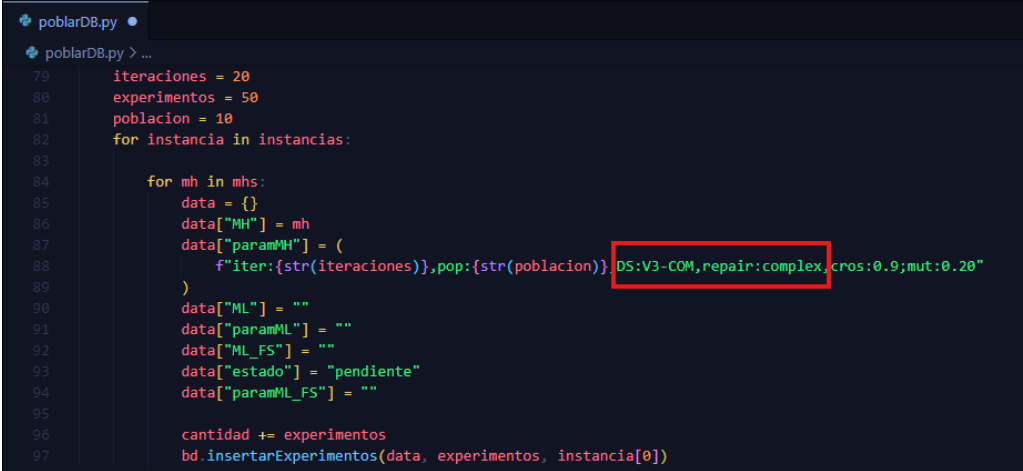
i.



```
78 iteraciones = 20
79 experimentos = 50
80 poblacion = 10
```

Ilustración 2. Ejemplo de configuración de ejecución.

- e. Indicar las preferencias en cuanto a discretización y reparación dentro de los parámetros de paramMH dentro del bucle for. Se incluye el siguiente ejemplo como referencia:
- i.



```
79 iteraciones = 20
80 experimentos = 50
81 poblacion = 10
82 for instancia in instancias:
83
84     for mh in mhs:
85         data = {}
86         data["MH"] = mh
87         data["paramMH"] = (
88             f"iter:{str(iteraciones)},pop:{str(poblacion)} DS:V3-COM,repair:complex,cros:0.9;mut:0.20"
89         )
90         data["ML"] = ""
91         data["paramML"] = ""
92         data["ML_FS"] = ""
93         data["estado"] = "pendiente"
94         data["paramML_FS"] = ""
95
96     cantidad += experimentos
97     bd.insertarExperimentos(data, experimentos, instancia[0])
```

Ilustración 3. Ejemplo de configuración de binarización y reparación.

- f. Guardar los cambios en el archivo.
- g. Dentro de la carpeta raíz del repositorio, ejecutar el script a través del siguiente comando en la terminal: **python3 poblarDB.py**
2. Ejecutar experimentos:
- a. A continuación, se debe llevar a cabo la ejecución de los experimentos con los que acabamos de poblar la base de datos, para ello, ejecutaremos el siguiente comando: **python3 main.py**
3. Análisis de resultados:
- a. Una vez ejecutados los experimentos, podremos obtener estadística relacionada a la información de ejecución. Este análisis se encuentra separado en dos: Análisis de resultados de fitness y análisis de resultados de tiempos de ejecución.
- b. Para ejecutar los análisis de resultados de fitness deberemos, desde la carpeta raíz del directorio, ejecutar el siguiente comando: **python3 AnalizarFitness.py**. Esto nos generará un archivo csv en la carpeta AnalisisFitness que contendrá información respecto a todas las instancias ejecutadas de AOA y GWO.
- c. Para ejecutar los análisis de resultados de tiempos de ejecución deberemos, desde la carpeta raíz del directorio, ejecutar el siguiente comando: **python3 AnalizarTiempos.py**. Esto nos generará dos archivos csv en la carpeta AnalisisTiempos que contendrá información respecto a los tiempos de ejecución respecto a las instancias (iteration_analysis) y experimentos (execution_time_analysis).
4. Generación de gráficos:
- a. Una vez ejecutados los experimentos, podremos generar gráficos relacionados a la información de ejecución. Estos gráficos se encuentran separado en dos: Gráficos de resultados de fitness y Gráficos de resultados de tiempos de ejecución. Los tipos de gráficos generados son: Gráfico de caja, gráficos de puntos y caja, gráficos de densidad, histogramas, gráficos de enjambre, gráficos de violín y gráficos de violín y puntos.
- b. Para generar los gráficos de resultados de fitness deberemos, desde la carpeta raíz del directorio, ejecutar el siguiente comando: **python3 GenerarGráficosFitness.py**. Esto generará los gráficos mencionados anteriormente dentro de la carpeta GraficosFitness que contendrán información respecto a todas las instancias ejecutadas de AOA y GWO.
- c. Para generar los gráficos de resultados de tiempos de ejecución deberemos, desde la carpeta raíz del directorio, ejecutar el siguiente comando: **python3 GenerarGráficosTiempos.py**. Esto

generará los gráficos mencionados anteriormente dentro de la carpeta GraficosTiempo que contendrán información respecto a todas las instancias ejecutadas de AOA y GWO.

Referencias

- Abualigah, L., Diabat, A., Mirjalilid, S., Abd Elazizf, M., & Gandomih, A. (2021). The Arithmetic Optimization Algorithm. Computer Methods in Applied Mechanics and Engineering.*
- Karp, R. M. (1972). Complexity of Computer Computations. Plenum Press.*
- Lanza-Gutierrez, J., Crawford, B., Soto, R., Berrios, N., Gomez-Pulido, J., & Paredes, F. (2017). Analyzing the Effects of Binarization Techniques when Solving the Set Covering Problem. Expert Systems with Applications.*
- Mirjalili, S., Mirjalili, S., & Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software.*