Manual de Usuario - Solver Set Covering Problem

Felipe Cisternas-Caneo[1]

Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
felipe.cisternas@pucv.cl

El presente manual hace referencia al código público disponible en el repositorio GitHub `https://github.com/FelipeCisternasCaneo/Solver_SCP.git`

1 Estructura del Sistema

Nuestro sistema está desarrollado en Python 3.10.5 64-bit y cuenta con 7 principales módulos los cuales son los siguientes:

- BD: Módulo donde se encuentra la base de datos y su respectiva gestión.
- Discretization: Módulo que está asociado a la binarización de metaheurísticas continuas para que puedan resolver problemas de optimización combinatoriales.
- Diversity: Módulo que estpa asociado para analizar el comportamiento de la población durante el proceso de optimización.
- Metaheuristics: Módulo donde se encuentran las metaheurísticas implementadas hasta la fecha.
- Problem: Módulo donde se encuentran los problemas de optimización a resolver. Acá se encuentra la lectura de instancias, cálculo de fitness y reparación de soluciones.
- Solver: Este módulo es el encargado de unir el módulo de metaheurísticas y el módulo de problema. Acá es donde se crea el ambiente para resolver una instancia de un problema.
- Análisis de resultados: En este módulo se realizan el análisis correspondiente a los experimentos realizados. Acá se pueden hacer tablas de resultados, gráficos de convergencia, gráficos de diversidad, gráficos de exploración-explotación, gráficos caja-bigote y gráficos de violines.

La Figura 1 nos muestra la estructura general de nuestro sistema.

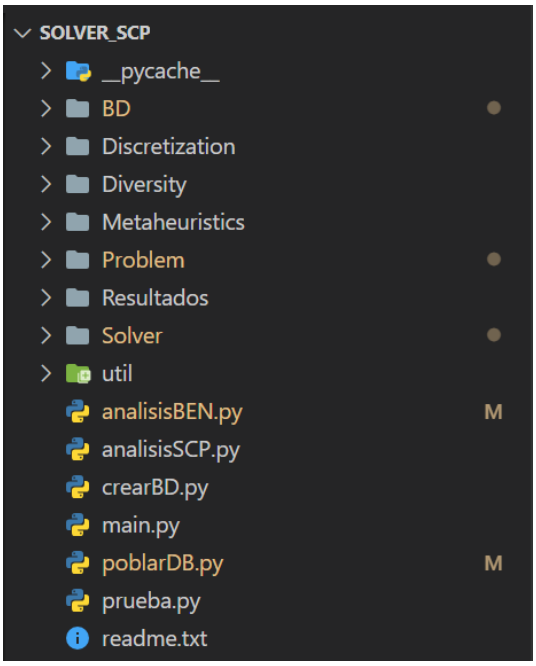

Fig. 1: Estructura del sistema

2 Base de datos

En nuestro sistema toda la información es almacenada en una base de datos relacional donde el motor de base de datos utilizado es el sqlite. Cuenta con 4 tablas las cuales son las siguientes:

- Instancias: En esta tabla se almacena los nombres de las instancias que se pueden ejecutar en nuestro sistema.
- Experimentos: En esta tabbla se almacena todos los experimentos pendientes, ejecutando y terminados. A esta tabla se consulta por experimentos pendientes.
- Resultados: Tal como su nombre lo indica, en esta tabla se almacena los resultados obtenidos en un experimento. Acá se almacena el fitness obtenido, el tiempo de ejecución del algoritmo y el vector solución.
- Iteraciones: En esta tabla se almacena un archivo en formato binario con todo el comportamiento de un experimento cuando realiza el proceso de optimización. Este archivo es consultado para realizar el análisis correspondiente.

La Figura 2 nos indica el modelo relacional del sistema.

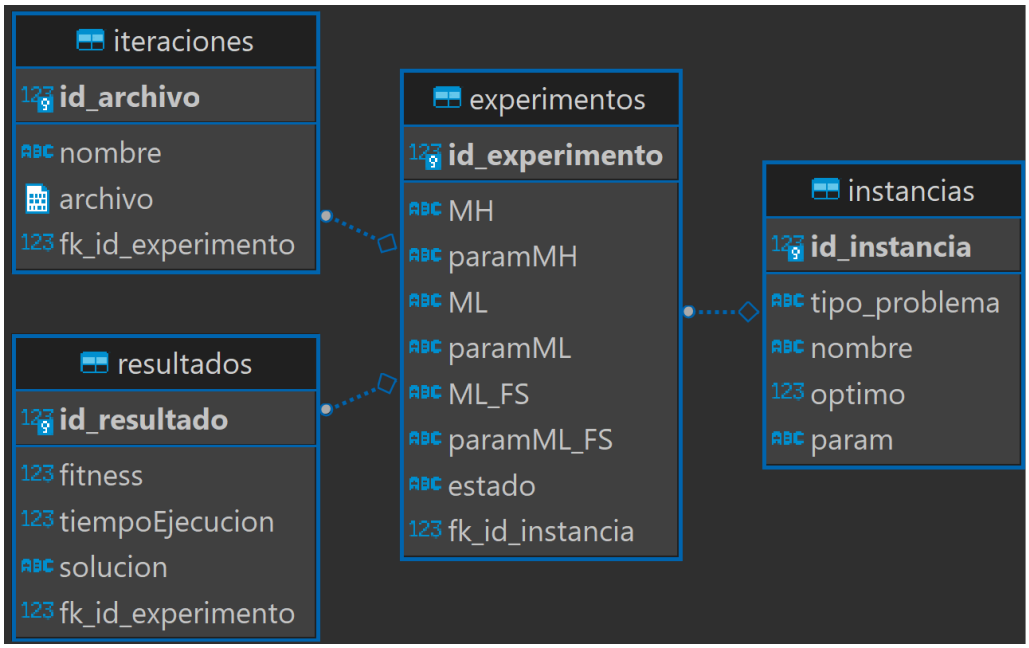

Fig. 2: Modelo Relacional Base de Datos

3 Metaheuristicas

Las metaheurísticas son algoritmos de proposito general que, con pocas modificaciones, son capaces de resolver diferentes problemas de optimización. El proceso de búsqueda consists en balancear la diversificación y la intensificación [13]. La diversificación o exploración tiene como objetivo encontrar regiones prometedoras del espacio de búsqueda y la intesificación o explotación consiste en intensificar la búsqueda e intentar encontrar mejores soluciones. A pesar que las metaheurísticas no garantizan el óptimo global, si entregan soluciones de alta calidad en tiempos razonables.

Las metaheurísticas tienen diferentes fuentes de inspiración como el comportamiento social de los humanos, la teoría de la evolución, comportamiento social de los animales o comportamiento físicos. Para mayor información sobre metaheurísticas recientes puede consultar en [12].

Hasta la fecha se encuentran disponibles las siguientes metaheurísticas:

- Genetic Algorithm: Una metaheurística insírada en las leyes de evolución de Darwin donde simula el intercambio genetico [10].
- Grey Wolf Optimizer: Una metaheurística inspirada en el comportamiento social jerárquico y de caza de los lobos grises [9].
- Moth-Flame Optimization: Una metaheurística inspirada en como las polillas se trasladan basandose en fuentes lumínicas [6].
- Pendulum Search Algorithm: Una metaheurística inspirada en el movimiento armónico del péndulo simple [1].
- Sine Cosine Algorithm: Una metaheurística inspirada en la dualidad de las funciones trigonométricas seno y coseno [7].
- Whale Optimization Algorithm: Una metaheurística inspirada en el comportamiento de caza de las ballenas jorobadas [8].

4 Discretización

La mayoría de las metaheurísticas existentes están diseñadas para resolver problemas de optimización continuos, por lo tanto, para resolver problemas de optimización combinatoriales binarios es necesario transferir las soluciones del dominio continuo real $\mathbb{R}$ al domibio discreto binario. En la literatura existen diferentes maneras de binarizar metaheurísticas donde las más utilizada es la Técnica de Dos pasos [2].

Tal como su nombre lo indica, el proceso de binarización se realiza en dos pasos. El primero paso consiste en transferir los números continuos de la metaheurística al dominio [0,1] aplicando una función de transferencia. Una vez transferido el número continuo, se aplica una regla de binarización el cuál nos termina por discretizar el número. La Figura 3 nos muestra el esquema general de la Técnica de Dos Pasos.

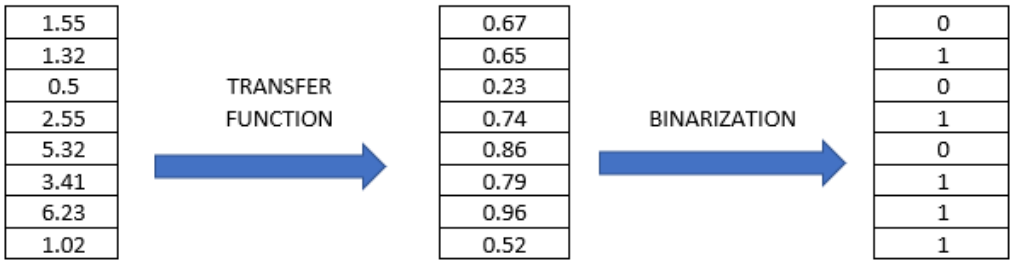

Fig. 3: Two-Step Technique

5 Diversidad

Uno de los apartados claves en las metaheurísticas es la exploración y explotación. La propuesta hecha en [11] nos presenta una forma de medir la exploración y explotación en terminos de porcentajes y calculándose mediante la diversidad poblacional. El porcentaje de exploración ($XPL\%$) y el porcentaje de explotación ($XPT\%$) se calculan de la siguiente manera:

$$XPL\% = \frac{Div}{Div_{max}} \cdot 100, \tag{1}$$

$$XPT\% = \frac{|Div - Div_{max}|}{Div_{max}} \cdot 100. \tag{2}$$

Donde Div representa la diversidad actual existente en la población y Div_{max} corresponde a la diversidad máxima registrada durante el proceso de optimización. En el presente sistema la diversidad utilizada es la propuesta por Hussain Kashif et. al. [4].

6 Problemas

6.1 Funciones matemáticas benchmarck

Estas problemas son los clásicos que utilizan los autores cuando presentan una metaheurística en sociedad. Son problemas matemáticos clasicos ampliamente utilizados en la literatura y se recomienda resolver estos problemas antes de saltar al Set Covering Problem. En el presente sistema se cuentan con 11 diferentes funciones matemáticas benchmark a resolver.

6.2 Set Covering Problem

El Set Covering Problem es un problema clásico de optimización combinatorial de la categoriá NP-Hard [3]. Este problema consiste en encontrar un conjunto de elements que perminta satisfacer un conjunto de necesidades al menor costo posible. Matemática la función objetivo es definida de la siguiente manera:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{3}$$

$$\min f(x) = \sum_{j=1}^{N} c_j x_j \tag{4}$$

Sujeto a las siguientes restricciones:

$$\begin{aligned} \sum_{j=1}^{N} a_{ij} x_j \geq 1 \quad \forall i \in I \\ x_j \in \{0,1\} \quad \forall j \in J \end{aligned} \tag{5}$$

Donde A es una matriz binario de tamaño m filas y n columnnas y $a_{ij} \in \{0,1\}$ es el valor de cada celda de la matriz A. El i y el j son las son las m filas y las n columnas respectivamente. Cuando una columna j satisface a una fila i tendremos un valor para a_{ij} es igual a 1, en caso contario tendrá un valor de 0. Adicionalmente, cada columna tiene asociado un costo $c \in C$, where $C = \{c_1, c_2, \dots, c_n\}$ junto con que $i = \{1, 2, \dots, m\}$ y $j = \{1, 2, \dots, n\}$ son el conjunto de filas y columnas respectivamente. Finalmente, x es un vector compuestos por unos y ceros, donde un uno nos la activación de una columna y un cero nos indica lo contrario. Para mayor información del modelo matemática del Set Covering Problem puede consultar en [5].

7 Solver

Acá se encontrará un solver para cada problema de optimización donde el pseudocódigo 1 nos indica el esquema general. La linea 1 hace referencia a la generación de soluciones aleatoria la cuál se realiza de acuerdo al problema de optimización a resolver donde para el Set Covering Problem será binaria. En la linea 2 se realiza la verificación de soluciones factibles donde acá verificamos que la solución cumpla con las restricciones del problema, en el caso del Set Covering, que una fila esté cubierta por al menos una columna. La linea 3 hace referencia a la reparación de soluciones en caso que ésta sea infactible. La linea 4 procedemos a calcular el fitness donde **solo se calcula el fitness a soluciones factibles**. La linea 5 hace referencia a la identificación de la mejor solución de la población. La linea 9 hace referencia a la pertubación de las soluciones donde Δ es implementado de diferentes maneras dependiendo de la metaheurística.

Algorithm 1 Esquema general de resolución

Entrada: Población $X = \{X_1, X_2, ..., X_i\}$
Salida: Población actualizada $X' = \{X'_1, X'_2, ..., X'_i\}$ y X_{best}

Inicialización aleatorio de la población X
Verificar factibilidad de cada individuo de la población
En caso que la solución se infactible, se repara
Evaluar la función objetivo en cada individuo de la población
Identificar al mejor individuo de la población (X_{best})
for *iteraciones* (t) **do**
    for *soluciones* (i) **do**
        for *dimensiones* (j) **do**
            $X_{i,d}^{t+1} = X_{i,d}^{t} + \Delta$
        end for
    end for
    Verificar factibilidad de cada individuo de la población
    Evaluar la función objetivo en cada individuo de la población
    En caso de detectar una nueva mejor solución, se actualiza X_{best}
end for
Retornar la población actualizada X donde X_{best} es la mejor solución

8 Ejecución de experimentos

Para ejecutar los experimentos primero debemos poblar la base de datos con experimentos. Para esto, debemos ejecutar el archivo "poblarDB.py". En este archivo indicamos que metaheurística vamos a utilizar, tamaño de población, cantidad de iteraciones, cantidad de veces que se realizará un experimento, que función de transferencia se aplicará y que regla de binarización se aplicará.

Una vez poblada la base de datos debemos ejecutar el archivo "main.py". Este archivo busca en la base de datos un experimento pendiente y lo comienza a ejecutar.

9 Dependencias

Para poder ejecutar este sistema en su computador debes tener las siguientes consideraciones:

- Instalar Python en su versión de 3.10.5 de 64-bit.
- Instalar la librería numpy en su versión 1.23.1
- Instalar la librería scipy en su versión 1.8.1
- Instalar la librería matplotlib en su versión 3.5.2
- Instalar la librería pandas en su versión 1.4.3
- Instalar la librería seaborn en su versión 0.11.2

References

1. Nor Azlina Ab. Aziz and Kamarulzaman Ab. Aziz. Pendulum search algorithm: An optimization algorithm based on simple harmonic motion and its application for a vaccine distribution problem. *Algorithms*, 15(6):214, 2022.
2. Marcelo Becerra-Rozas, José Lemus-Romani, Felipe Cisternas-Caneo, Broderick Crawford, Ricardo Soto, Gino Astorga, Carlos Castro, and José García. Continuous metaheuristics for binary optimization problems: An updated systematic literature review. *Mathematics*, 11(1):129, 2022.
3. Michael R Garey and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.
4. Kashif Hussain, William Zhu, and Mohd Najib Mohd Salleh. Long-term memory harris' hawk optimization for high dimensional and optimal power flow problems. *IEEE Access*, 7:147596–147616, 2019.
5. Jose M Lanza-Gutierrez, NC Caballe, Broderick Crawford, Ricardo Soto, Juan A Gomez-Pulido, and Fernando Paredes. Exploring further advantages in an alternative formulation for the set covering problem. *Mathematical Problems in Engineering*, 2020, 2020.
6. Seyedali Mirjalili. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-based systems*, 89:228–249, 2015.

7. Seyedali Mirjalili. Sca: a sine cosine algorithm for solving optimization problems. *Knowledge-based systems*, 96:120–133, 2016.
8. Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in engineering software*, 95:51–67, 2016.
9. Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.
10. Seyedali Mirjalili and Seyedali Mirjalili. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, pages 43–55, 2019.
11. Bernardo Morales-Castañeda, Daniel Zaldívar, Erik Cuevas, Fernando Fausto, and Alma Rodríguez. A better balance in metaheuristic algorithms: Does it exist? *Swarm and Evolutionary Computation*, 54:100671, 2020.
12. Kanchan Rajwar, Kusum Deep, and Swagatam Das. An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review*, pages 1–71, 2023.
13. El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.