Jaya y Lobo Gris aplicado a instancias de QAP

Alhely González Luna[1], Rogelio González Velazquez[2],
Abraham Sánchez López [3], Erika Granillo Martinez [4]

[1]FCCO BUAP, gl224470257@alm.buap.mx
[2]FCCO BUAP,rogelio.gonzalez@correo.buap.mx
[3]FCCO BUAP, abraham.sanchez@correo.buap.mx
[4]FCCO BUAP, erika.granillom@correo.buap.mx

Resumen

El problema de asignación cuadrática (QAP), por su complejidad NP-hard y su presencia en múltiples aplicaciones industriales, sigue siendo un tema clave en ciencias computacionales. Cuando se enfrentan instancias grandes y recursos limitados, las metaheurísticas ofrecen soluciones viables. Métodos como Búsqueda Tabú, Recocido Simulado, Algoritmo Genético y PSO han mostrado buenos resultados en la resolución de QAP. Sin embargo, metaheurísticas más recientes como Jaya (2016) y Lobo Gris (2011) han sido poco exploradas en este contexto. En este trabajo, se presenta una comparación de 9 metaheurísticas aplicadas a instancias representativas de la biblioteca QAPLIB.

Palabras clave: QAP, Metaheurísticas, Jaya, Lobo Gris, PSO, GA, TS, SA.

1. Introducción

El problema de asignar instalaciones a ubicaciones de tal forma que cada ubicación esté asignada a una sola ubicación y viceversa, en donde se conocen las distancias entre ubicaciones y las demandas de flujo entre instalaciones se conoce como el Problema de Asignación Cuadrática (Quadratic Assignment Problem), este problema pertenece a la clase NP-Hard [8].

Algunas aplicaciones importantes del QAP en la industria son: diseños hospitalarios, cableado de tablero, modificación de campus, arqueología, quiemica de reacciones, entre muchas más [5].

Al ser un problema NP-hard, no es posible encontrar una solución exacta en tiempo polinomial, existen varios métodos para hallar soluciones dependiento del tamaño de la instancia (ubicaciones e instalaciones). Para problemas pequeños es posible hallar soluciones mediante algorítmos de Branch and Bound, Métodos de Relajación o Programación Dinámica. Sin embargo, para instancias de gran tamaño, no es posible aplicar estos métodos exactos ya que se requieren demasiados recursos computacionales, en estos casos, se aplican algorítmos metaheurísiticos que usualmente iteran sobre posibles soluciones aleatorias hasta encontrar una solución óptima de entre todas las posibles mediante reglas de

optimización; algunas de las metaheurísticas más populares son : Búsqueda Tabú, Recocido Simulado, GRASP, Algorítmos Genéticos, o metaheurísticas basadas en poblaciones como hormigas, abejas, etc [2].

Dos metaheurísticas relativamente recientes son, Jaya [7] y Lobo Gris [33] publicadas en 2016 y 2011 respectivamente, han tenido buenos resultados en localización de máximos y mínimos de funciones. Ambas tienen como parámetros el número de iteraciones y el tamaño de la población, aunque difieren en la técnica de optimización, sin embargo, al tener pocos parámetros y sencilla codificación representan una buena elección para explorar nuevas metaheurísticas aplicadas al QAP.

En el siguiente trabajo comparamos el resultado de 9 algoritmos, incluidos Jaya y Lobo Gris para las instancias que se encuentran en qaplib. El documento está organizado de la siguiente forma: En la sección dos se presenta una revisión teórica del problema de asignación cuadrática, en la sección tres presentaremos un conjunto de algoritmos para un benchmark que han sido previamente probados, en la sección cuatro revisaremos la teoría detrás de Jaya y Lobo Gris, destacaremos sus principales diferencias y ventajas, en la sección cinco mostraremos los resultados obtenidos para algunas instancias de prueba y en la sección seis daremos las conclusiones de este trabajo.

2. El problema de asignación cuadrática (QAP)

Los problemas del mundo real pueden dividirse en problemas de decisión y problemas de optimización. Cualquier problema cuya respuesta sea 1 o 0 es un problema de decisión. Por otro lado, si el problema implica la identificación de un óptimo (mínimos o máximos) de una función de costo se trata de un problema de optimización. [5].

El problema de asignación cuadrática (QAP) introducido por primera vez por Koopmans y Beckmann [1], consiste en asignar un conjunto n de instalaciones a un conjunto de n ubicaciones, donde el flujo entre instalaciones y la distancia entre ubicaciones se conoce previamente [2]. El problema puede formularse matemáticamente como:

$$\min_{p \in P} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)} \tag{1}$$

Donde f es la matriz de flujo y d la matriz de distancias, ambas de tamaño $n \times n$, y p es una permutación de tamaño n.

Por ejemplo, para $n = 3$, una permutación p podría ser:

$$p = [3, 1, 2]$$

interpretable como que la instalación 3 se asigna a la ubicación 1, la instalación 1 a la ubicación 2 y la instalación 2 a la ubicación 3.

El conjunto de todas las permutaciones posibles se denota con P. Para cada par de asignaciones (instalación i -ubicación j) el costo asociado al par será el flujo f_{ij} multiplicado por la distancia d_{ij}. La suma de estos términos sobre todos los pares dentro de P representan el costo total para la permutación p, mostrado en (1). El objetivo es encontrar una permutación p^* en P que minimice el costo total [3]. El problema de QAP puede verse como un problema de optimización combinatoria en el que cada permutación tiene asociado un costo, en este caso flujo por distancia, y debe hallarse la permutación que de el mejor costo.

Se demostró que el QAP pertenece a la clase de problemas NP-hard [4], por lo que no existe aún un algoritmo que tenga un error relativo garantizado menor que una constante para cada instancia del QAP salvo cuando P = NP [9].

Para instancias de tamaño mayor a 30 [4] no es posible utilizar un método exacto como Branch and Bound debido a las restricciones computacionales, ha surgido la necesidad de algoritmos como las metaheurísticas que están diseñadas para hallar un óptimo (casi siempre global) comenzando con un espacio de soluciones generado de manera aleatoria y optimizado mediante emulaciones de comportamientos observados en la naturaleza o en nuestra sociedad, como lo son; recocido simulado, colonia de hormigas, colonia de abejas, teacher learning, PSO, etc.

Una de las ventajas de utilizar una metaheurística para el problema del QAP es que pueden probarse instancias de mayor tamaño sin necesidad de sacrificar una buena aproximación, debido a la naturaleza misma de la metaheuristica. Sin embargo, tienen la desventaja de que dependen de la solución inicial en la que se comienza la optimización asi como de los parámetros de la misma. Es por ello que, metaheurísticas con pocos parámetros como Jaya y Lobo gris representan opciones atractivas para probar en el ámbito del QAP.

3. Revisión de metaheurísticas clásicas

En esta sección se muestra un resumen de los algoritmos que compararemos en este artículo, un poco de su funcionamiento, ventajas, desventajas y resultados que servirán como métricas de referencia para los nuevos algoritmos introducidos en este artículo.

3.1. Fast Ant System (FANT)

3.1.1. Ant Colony Optimization (ACO)

La Optimización por Colonia de Hormigas (*Ant Colony Optimization*, ACO) es una metaheurística que ha sido aplicada exitosamente para resolver diversos problemas de optimización combinatoria [10].

En ACO, varias generaciones de hormigas artificiales buscan buenas soluciones. Cada hormiga de una generación construye una solución paso a paso, tomando varias decisiones hasta encontrar una solución. De forma similar a las hormigas reales, las hormigas artificiales que encuentran una buena solución marcan sus caminos a través del espacio de decisiones depositando cierta cantidad de feromona sobre las aristas del camino. Las siguientes hormigas de la próxima generación se sienten atraídas por la feromona, por lo que buscarán en el espacio de soluciones cercano a las buenas soluciones. Una de las desventajas de numerosos marcos basados en hormigas artificiales es su gran número de parámetros y la dificultad para ajustarlos.

3.1.2. FANT

FANT se aparta del esquema clásico de ACO (Ant Colony Optimization) mediante varias innovaciones fundamentales. Primero, implementa una *actualización inmediata de feromonas*, aplicando refuerzo tan pronto una hormiga completa su solución, lo que favorece una rápida convergencia al aprovechar caminos prometedores [11, 12]. Segundo,

adopta un modelo centralizado donde una reina coordina las hormigas y gestiona una memoria compartida de feromonas, aportando una estructura más organizada que el ACO tradicional [11]. Además, FANT combina explícitamente *mecanismos de intensificación y diversificación*, implementando una evaporación constante junto a actualizaciones rápidas de refuerzo [12]. Emplea también *listas de candidatos* para reducir la complejidad computacional durante la construcción de soluciones [12].

Una implementación lista y probada del algoritmo FANT para instancias del QAP puede econtrarse en FANT de Taillard.

3.2. Búsqueda Tabú (TS)

Las ideas principales de la Búsqueda Tabú (*Tabu Search*, TS) pueden resumirse brevemente de la siguiente manera. El primer ingrediente (común a la mayoría de los procedimientos heurísticos y algorítmicos) es definir un **vecindario**, o un conjunto de movimientos que pueden aplicarse a una solución dada para producir una nueva [9].

Entre todas las soluciones vecinas, TS busca aquella con la mejor evaluación de la función de costo. En el caso más simple, dicha evaluación dicta la elección del movimiento que más mejora la función objetivo. Si no existen movimientos que mejoren la solución (lo que indica una especie de óptimo local), TS elige aquel que menos degrade la función objetivo[9]. Para evitar regresar al óptimo local recién visitado, el movimiento inverso debe ser prohibido. Esto se logra almacenando dicho movimiento —o más precisamente, una caracterización del mismo— en una estructura de datos que comúnmente se gestiona como una lista circular y se denomina **lista tabú**. Esta lista contiene un número s de elementos que definen los movimientos prohibidos (tabú); el parámetro s se conoce como el **tamaño de la lista tabú**[9]. Sin embargo, la lista tabú puede prohibir ciertos movimientos relevantes o interesantes, como aquellos que conducen a una solución mejor que la mejor encontrada hasta el momento. Por ello, se introduce un **criterio de aspiración**, el cual permite seleccionar movimientos tabú si se considera que son prometedores o beneficiosos [9].

En el siguiente enlace pueden encontrarse dos algoritmos escritos en c que implementan búsqueda tabú, se muestran dos algoritmos *new* y *old*, implementados en c y c++ respectivamente, Tabú Search New C, Tabú Search Old C++ .

3.3. Recocido Simulado (SA)

La metaheurística de recocido simulado está basada en el proceso de la metalurgia, en el que los metales se calientan y enfrian lentamente para remover impurezas del mismo. Introducido por primera vez en 1983 en [15], y es aplicado en la optimización combinatoria debido a su similitud con la mecánica estadística de sistemas complejos.

La idea central de este método es que ciertos movimientos hacia soluciones peores pueden ser necesarios para evitar que un esquema de optimización quede atrapado en un óptimo local pobre. Para un problema en particular, se debe especificar una **topología** (o estructura de vecindad, por ejemplo 2-intercambio, 3-intercambio, etc.), así como un método eficiente para moverse de una solución a una vecina y calcular el cambio resultante en la función objetivo. En general, las mejoras siempre se aceptan, mientras que los movimientos hacia soluciones peores se aceptan con una probabilidad que depende del tamaño del incremento y del estado de varios parámetros de control. En el siguiente enlace

se encuentra un código que implementa el recocido simulado basado en [14] implementado por E. Taillard [11] Recocido Simulado QAP.

3.4. Optimización de Enjambre de Partículas (PSO)

La Optimización por Enjambre de Partículas (*Particle Swarm Optimization*, PSO) tiene sus raíces en dos metodologías principales. Quizá la más evidente es su relación con la vida artificial (*A-life*) en general, y en particular con el comportamiento de bandadas de aves, bancos de peces y teorías de enjambres. Sin embargo, también está relacionada con la computación evolutiva, y guarda vínculos tanto con los algoritmos genéticos como con la programación evolutiva. Estas relaciones se revisan brevemente en el artículo original[17].

La PSO, tal como fue desarrollada por los autores, se basa en un concepto muy simple, y sus paradigmas pueden implementarse en unas cuantas líneas de código. Solo requiere operadores matemáticos primitivos y es computacionalmente económica, tanto en cuanto al uso de memoria como en velocidad de ejecución.

Un código que implementa el PSO en python para el problema de QAP puede encontrarse en Mostapha Kalami Heris

3.5. Algoritmo genético (GA)

El Algoritmo Genético (*Genetic Algorithm*, GA) puede considerarse un método para optimizar la búsqueda de soluciones en problemas difíciles, basado en el principio de selección genética. Además de ser una herramienta de optimización, también se utiliza en el aprendizaje automático y en investigación y desarrollo.

Es análogo a los procesos biológicos de generación de cromosomas, en donde se aplican operadores genéticos como la selección, el cruce (*crossover*) y la mutación sobre una población aleatoria inicial. El objetivo del GA es generar soluciones en generaciones sucesivas. El grado de éxito en la producción de individuos está directamente relacionado con la aptitud de la solución que representan, asegurando así que la calidad de las soluciones mejore generación tras generación [30].

El proceso concluye cuando se alcanza una solución satisfactoria para el problema que se desea optimizar, típicamente asociado a un sistema computacional evaluable. John Holland es considerado el padre fundador del algoritmo genético original, desarrollado en la década de 1970. Además, este enfoque puede verse como una extensión de la idea de búsqueda aleatoria en un espacio definido, como la representada por Charles Darwin, para resolver problemas de forma eficiente.

Un código que implementa el GA para el problema de QAP puede encontrarse en Mostapha Kalami Heris

3.6. Luciérnagas (Firefly)

Las luciérnagas son capaces de producir luz gracias a órganos fotogénicos especiales situados muy cerca de la superficie del cuerpo, detrás de una ventana de cutícula translúcida [21]. Se sabe que las señales bioluminiscentes cumplen funciones como parte de rituales de cortejo, métodos de atracción de presas, orientación social o como señal de advertencia para los depredadores[22] [23, 30]. El Algoritmo de Luciérnagas, desarrollado recientemente por Xin-She Yang en la Universidad de Cambridge y presentado en el Capítulo 8 de su monografía [27, 30] parte del supuesto de que la solución de un problema de

optimización puede percibirse como un agente (luciérnaga) que "brilla"proporcionalmente a su calidad dentro del entorno del problema considerado. Como consecuencia, cada luciérnaga más brillante atrae a sus compañeras, lo cual permite explorar el espacio de búsqueda de manera más eficiente. Entre otras metaheurísticas inspiradas en la naturaleza con un enfoque similar se encuentran la Optimización por Enjambre de Partículas (PSO) [28] y la técnica de optimización por Colonia de Abejas Artificiales (ABC) [29, 30].

4. Jaya y Lobo Gris

Todos los algoritmos evolutivos y los basados en inteligencia de enjambre son algoritmos probabilísticos que requieren parámetros de control comunes como el tamaño de población, el número de generaciones, el tamaño de la élite, entre otros. Además de estos parámetros generales, cada algoritmo necesita también parámetros de control específicos según su naturaleza [7].

Por ejemplo:

- El Algoritmo Genético (GA) utiliza la probabilidad de mutación, la probabilidad de cruce y un operador de selección.
- La Optimización por Enjambre de Partículas (PSO) utiliza el peso de inercia, y parámetros sociales y cognitivos.
- La Optimización por Colonia de Abejas Artificiales (ABC) requiere el número de abejas observadoras, abejas empleadas, abejas exploradoras y un límite de intentos.
- El Algoritmo de Búsqueda Armoniosa (HS) usa la tasa de consideración de memoria armónica, la tasa de ajuste de tono y el número de improvisaciones.

4.1. Jaya

De forma similar, otros algoritmos como Estrategias Evolutivas (ES), Programación Evolutiva (EP), Diferential Evolution (DE), Optimización por Colonia de Hormigas (ACO), entre otros, requieren también una afinación precisa de sus parámetros específicos [7].

La correcta sintonización de estos parámetros específicos es un factor crucial que afecta significativamente el desempeño de los algoritmos mencionados. Una mala elección puede incrementar el esfuerzo computacional o conducir a soluciones subóptimas (óptimos locales)[7].

Considerando este problema, Rao et al. (2011) [32] introdujeron el algoritmo de Optimización Basado en Enseñanza-Aprendizaje (TLBO, por sus siglas en inglés), el cual no requiere ningún parámetro específico del algoritmo. El algoritmo TLBO solamente necesita los parámetros comunes como el tamaño de la población y el número de generaciones para su funcionamiento.

El algoritmo TLBO ha ganado una amplia aceptación entre los investigadores en optimización [31, 32].

4.1.1. Algoritmo

Sea $f(x)$ la función objetivo que se desea minimizar (o maximizar), el algoritmo propuesto funciona de la siguiente forma [7]:

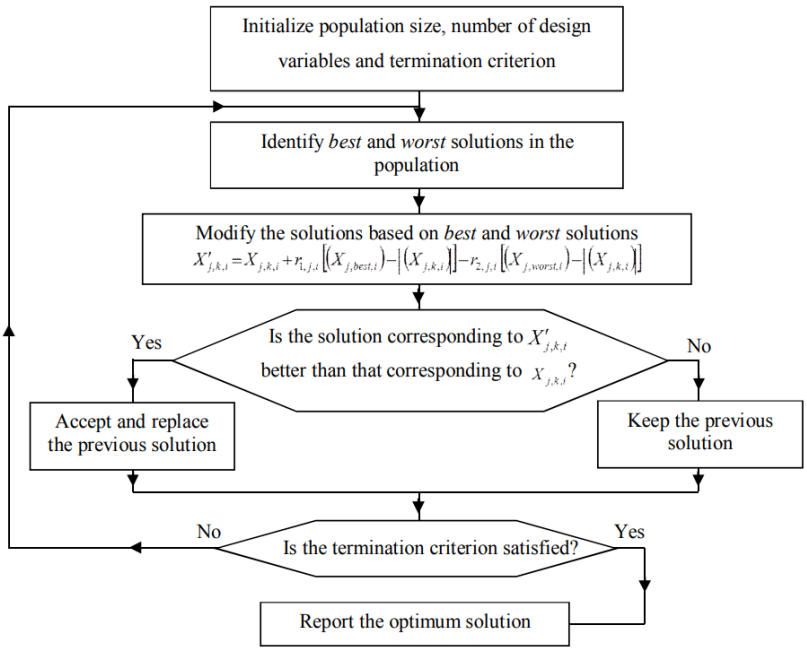

Figura 1: Flujo del algoritmo Jaya [7]

- En la iteración i, se tiene:
  - m: número de variables de diseño ($j = 1, 2, \ldots, m$).
  - n: número de soluciones candidatas o tamaño de población ($k = 1, 2, \ldots, n$).
- Se identifican las soluciones:
  - **Mejor** (best) con valor $f(x)_{\text{best}}$.
  - **Peor** (worst) con valor $f(x)_{\text{worst}}$.
- Si $X_{j,k,i}$ es el valor de la j-ésima variable de la k-ésima solución en la iteración i, su valor actualizado $X'_{j,k,i}$ se calcula como:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}(X_{j,\text{best},i} - |X_{j,k,i}|) - r_{2,j,i}(X_{j,\text{worst},i} - |X_{j,k,i}|)$$

- Donde:
  - $r_{1,j,i}, r_{2,j,i}$ son números aleatorios en el rango $[0, 1]$.
  - El primer término promueve el acercamiento a la mejor solución.
  - El segundo término evita acercarse a la peor solución.
- La nueva solución $X'_{j,k,i}$ se acepta sólo si mejora el valor de la función objetivo.
- Las soluciones aceptadas al final de cada iteración se conservan y se utilizan como base para la siguiente iteración.

4.2. Lobo Gris (GW)

Los lobos grises del norte, la especie más grande del género Canis, son reconocidos por presentar algunos de los comportamientos sociales intraespecíficos más complejos entre los carnívoros [34, 6]. Los autores resumen estos comportamientos, que incluyen vivir en unidades sociales llamadas manadas, cazar en grupo, participar en el cuidado colectivo de las crías y en la defensa grupal del alimento y el territorio. Vivir y forrajear en manada es más común donde las presas principales son ungulados grandes, donde la pérdida de alimento por carroñeros es alta y donde la defensa territorial es esencial [35, 36, 37, 6].

Los lobos cazan grandes ungulados, por ejemplo, alces (Alces alces) y bisontes (Bison bison), en manadas de dos o más individuos [38]. Se ha sugerido que la cooperación en manadas podría aumentar la eficiencia individual en la caza de presas grandes, pero la gran mayoría de los datos indican que el beneficio por individuo en la caza social es limitado [39, 40, 6].

También se sugiere que estas unidades familiares son jerárquicas [41, 6] y que la cooperación social entre individuos indica una inteligencia avanzada que apoya la planificación estratégica de las cacerías [41]. Estas afirmaciones suelen basarse en la observación de dos tipos de comportamientos de caza: emboscadas y persecuciones en relevo [39, 42].

El algoritmo representa un modelo computacional multiagente en el cual los lobos y las presas son representados por agentes que obedecen reglas simples y descentralizadas. Usando los resultados de simulaciones computacionales, describimos el comportamiento grupal producido por la combinación de estas reglas aplicadas individualmente, y observamos la emergencia de patrones de comportamiento colectivo [6].

4.2.1. Algoritmo

Para comprender cómo el GWO es capaz de resolver problemas de optimización, se pueden considerar los siguientes puntos [33]:

- La jerarquía social propuesta permite que el GWO conserve las mejores soluciones encontradas durante las iteraciones.
- El mecanismo de cercado define una vecindad con forma de círculo alrededor de las soluciones, que puede extenderse a dimensiones superiores como una hiperesfera.
- Los parámetros aleatorios A y C permiten que las soluciones candidatas tengan hiperesferas con radios aleatorios distintos.
- El método de caza propuesto ayuda a que las soluciones candidatas localicen la posición probable de la presa.
- La exploración y la explotación están garantizadas por los valores adaptativos de a y A.
- Estos valores adaptativos permiten una transición suave entre las fases de exploración y explotación.
- A medida que A disminuye, la mitad inicial de las iteraciones se dedica a la exploración ($|A| \geq 1$), y la mitad final a la explotación ($|A| < 1$).
- El algoritmo GWO solo requiere ajustar dos parámetros principales: a y C.

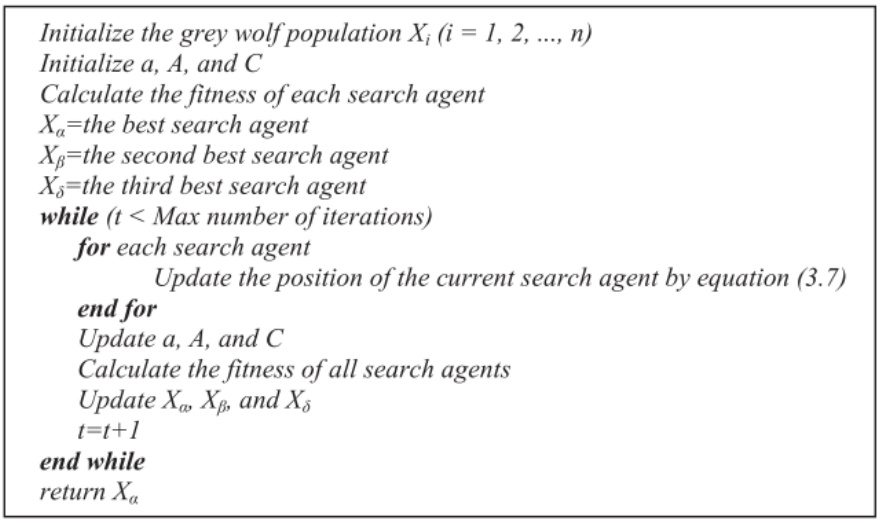

Figura 2: Pseudocódigo que implementa el algoritmo de Lobo Gris [33]

5. Comparación en instancias de QAP

En esta sección mostraremos los resultados de las metaheurísticas antes descritas para algunas instancias de tamaño mayor a 30 disponibles en qaplib, para dichas instancias se conoce el mejor valor obtenido hasta ahora, con lo cual calcularemos el GAP con cada una de las metaheurísticas. Para tener una comparación significativa fijaremos el número de iteraciones a 500 para todas las metaheurísticas y usaremos los valores sugeridos en la literatura para los parámetros específicos de cada una.

Se desarrolló un programa en python con una interfaz de usuario que permite:

1. Descargar en el directorio de trabajo todas las instancias que se encuentan en instancias completas con sus respectivos archivos de solución.

2. Permite al usuario elegir de entre las 9 metaheurísticas aqui revisadas, así como fijar los parámetros correspondientres de cada una.

3. Genera un archivo csv con los resultados obtenidos después de completar el número de iteraciones, el archivo csv contiene, la mejor solución, el costo asociado a esta, el gap con la mejor solución conocida y el tiempo de ejecución en CPU en segundos (Windows 11 en un procesador 12th Gen Intel(R) Core(TM) i5-12450HX)

5.1. Implementacion de metaheurísticas

5.1.1. FANT, TS, SA

Los códigos para FANT, TS y SA fueron tomados de Taillard, sin embargo los códigos originales están escritos en c y c++, para convertirlos a python se hizo uso del LLM ChatGPT en su versión gratuita utilizando la función de razonamiento con el siguiente prompt *Escribe el siguiente codigo de c a python manteniendo fielmente la estructura, herramientas y optimización <introducir códigos en c o c++>*.

5.1.2. PSO, GA, Firefly

Los códigos para PSO, GA, Firefly fueron tomados de yarpiz, sin embargo los códigos originales están escritos en matlab, para convertirlos a python se hizo uso del LLM ChatGPT en su versión gratuita utilizando la función de razonamiento con el siguiente prompt *Escribe el siguiente codigo de matlab a python manteniendo fielmente la estructura, herramientas y optimización <introducir códigos en formato matlab>*.

5.2. Jaya y Lobo Gris

Estas dos implementaciones fueron escritas en python siguiendo los pseudocódigos asociados a cada artículo.

Se construyó una interfaz básica con la librería tkinter y se utilizó PyInstaller para generar el archivo ejecutable.

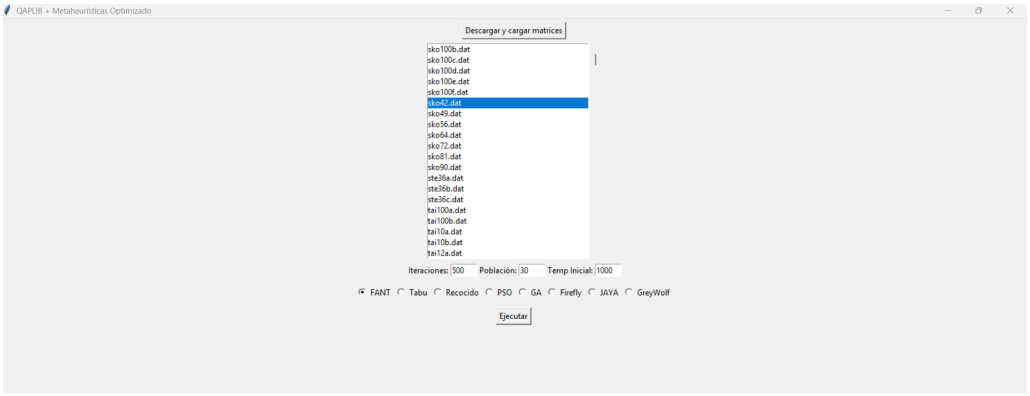

Figura 3: Interfaz de Usuario del programa propuesto

6. Resultados

En la tabla **??** se muestran los mejores valores obtenidos para cada instancia seleccionada, así como el GAP del resultado hallado con el mejor conocido

7. Conclusiones

Las conclusiones deben resumir los hallazgos principales, reflexionar sobre limitaciones y sugerir líneas de trabajo futuro.

Referencias

[1] T. Koopmans y M. Beckmann. *Assignment problems and the location of economic activities.* Econometrica, **25**(1):53–76, 1957.

[2] R. P. Ch, O. G. Hurtado y J. Moncada. *Exact And Approximate Sequential Methods In Solving The Quadratic Assignment Problem.* Journal of Language and Linguistic Studies, **18**(3):237–244, 2022.

[3] James, T., Rego, C., y Glover, F. (2009). "Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem", *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 39, pp. 579–596.

[4] Sahni, S. y Gonzalez, T. (1976). "P-complete approximation problems", *Journal of the Association for Computing Machinery*, vol. 23, pp. 555–565.

[5] Bhati, R.K. y Rasool, A. (2014). "Quadratic Assignment Problem and its Relevance to the Real World: A Survey", *International Journal of Computer Applications*, vol.96, no.9, pp.42–47. ISSN-0975-8887. DOI: 10.5120/16825-6584.

[6] Muro, C., Escobedo, R., Spector, L., y Coppinger, R.P. (2011). "Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations", *Behavioural Processes*, vol.88, no.3, pp.192–197. DOI10.1016/j.beproc.2011.09.006.

[7] Rao, R.V. (2016). "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems", *International Journal of Industrial Engineering Computations*, vol.7, no.1, pp.19–34. DOI 10.5267/j.ijiec.2015.8.004.

[8] Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., y Querido, T. (2007). "A survey for the quadratic assignment problem", *European Journal of Operational Research*, vol.176, no.2, pp.657–690. doi:10.1016/j.ejor.2005.09.032.

[9] Taillard, É.D. (1991). "Robust tabu search for the quadratic assignment problem", *Parallel Computing*, vol.17, no.4–5, pp.443–455. doi:10.1016/S0167-8191(05)80147-4.

[10] Merkle, D. y Middendorf, M. (2002). "Fast Ant Colony Optimization on Runtime Reconfigurable Processor Arrays", *Genetic Programming and Evolvable Machines*, vol.3, no.4, pp.345–361. doi:10.1023/A:1020936909085.

[11] E.D. Taillard, "FANT: Fast Ant System," IDSIA Technical Report 46-98, Lugano, Switzerland, Oct.15,1998. Informe original que presenta FANT, definiéndolo como un sistema rápido de hormigas con actualización inmediata de feromonas, coordinación central ("Queen") y estrategias de intensificación/diversificación

[12] "Fast Ant System," Algorithm Afternoon, 2025. Reseña sobre FANT que resalta tres innovaciones clave: actualización inmediata, evaporación constante y uso de listas de candidatos para ganar eficiencia

[13] "Fast Ant System (FANT)," en Ant Colony Optimization, pp.42–54. Descripción técnica de FANT que señala que usa una única hormiga (en lugar de colonia), actualizaciones

simplificadas de feromonas sin evaporación continua, y listas candidatas como mecanismo de reducción de complejidad

[14] Connolly, D.T. (1990). "An improved annealing scheme for the QAP", *European Journal of Operational Research*, vol.46, no.1, pp.93–100. DOI10.1016/0377-2217(90)90301-Q.

[15] Kirkpatrick, S., Gelatt, C.D. Jr., y Vecchi, M.P. (1983). "Optimization by simulated annealing", *Science*, vol.220, no.4598, pp.671–680. doi:10.1126/science.220.4598.671.

[16] Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

[17] Kennedy, J. y Eberhart, R.C. (1995). "Particle Swarm Optimization", en *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, pp.1942–1948. doi:10.1109/ICNN.1995.488968.

[18] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

[19] Lambora, A., Gupta, K., Chopra, K. (2019). "Genetic Algorithm–A Literature Review", in *Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pp.380–384. doi:10.1109/COMITCon.2019.8862255.

[20] Lewis, S.M. Cratsley, C.K. (2008). Flash signal evolution, mate choice, and predation in fireflies. *Annual Review of Entomology*, 53, 293–321.

[21] Lall, A.B., et al. (2000). Circadian control of luciferase mRNA accumulation in the firefly. *Insect Biochemistry and Molecular Biology*, 30(3), 191–196.

[22] Viviani, V.R. (2002). The origin, diversity, and structure function relationships of insect luciferases. *Cellular and Molecular Life Sciences*, 59, 1833–1850.

[23] Buck, J., Buck, E. (1966). Biology of synchronous flashing of fireflies. *Nature*, 211(5050), 562.

[24] Kim, J., et al. (2005). Firefly-inspired synchronization for wireless sensor networks. *ACM SIGCOMM*.

[25] Yang, X.S. (2009). Firefly algorithms for multimodal optimization. *Stochastic Algorithms: Foundations and Applications (SAGA 2009)*.

[26] Shen, H., Wang, Y. (2007). Firefly-based multi-robot cooperation control. *IEEE International Conference on Robotics and Automation*.

[27] Yang, X.S. (2009). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.

[28] Kennedy, J., Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*, 4, 1942–1948.

[29] Karaboga, D., Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: Artificial Bee Colony (ABC) algorithm. *Journal of Global Optimization*, 39(3), 459–471.

[30] Lambora, A., Gupta, K., Chopra, K. (2019). "Genetic Algorithm–A Literature Review", in *Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pp.380–384. doi:10.1109/COMITCon.2019.8862255.

[31] Rao, R. V., Savsani, V. J., Vakharia, D. P. (2011). Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3), 303–315.

[32] Rao, R. V. (2015). *Teaching-Learning-Based Optimization Algorithm: And Its Engineering Applications*. Springer.

[33] Mirjalili, S., Mirjalili, S.M., y Lewis, A. (2014). "Grey Wolf Optimizer", *Advances in Engineering Software*, vol.69, pp.46–61. doi:10.1016/j.advengsoft.2013.12.007.

[34] Macdonald, D. W., et al. (2004). *The New Encyclopedia of Mammals*. Oxford University Press.

[35] Mech, L. D. (1999). Alpha status, dominance, and division of labor in wolf packs. *Canadian Journal of Zoology*, 77(8), 1196–1203.

[36] Fuller, T. K., Mech, L. D., & Cochrane, J. F. (2003). Wolf population dynamics. In L. D. Mech & L. Boitani (Eds.), *Wolves: Behavior, Ecology, and Conservation*. University of Chicago Press.

[37] Vucetich, J. A., Peterson, R. O., & Waite, T. A. (2004). Raven scavenging favors group foraging in wolves. *Animal Behaviour*, 67(6), 1117–1126.

[38] Mech, L. D., & Boitani, L. (Eds.) (2003). *Wolves: Behavior, Ecology, and Conservation*. University of Chicago Press.

[39] Peterson, R. O., & Ciucci, P. (2003). The wolf as a carnivore. In L. D. Mech & L. Boitani (Eds.), *Wolves: Behavior, Ecology, and Conservation*. University of Chicago Press.

[40] Thurber, J. M., & Peterson, R. O. (1993). Effects of population density and pack size on the foraging ecology of gray wolves. *Journal of Mammalogy*, 74(4), 879–889.

[41] Packard, J. M. (2003). Wolf behavior: reproductive, social, and intelligent. In L. D. Mech & L. Boitani (Eds.), *Wolves: Behavior, Ecology, and Conservation*. University of Chicago Press.

[42] Mech, L. D. (2007). *Handbook of Wolf Ecology*. International Wolf Center.