El Problema del viajante desde algunos Algoritmos Metaheurísticos

Realizado por

Kevin Alberto Montañez Marquez

Juan Esteban Muñoz Rodríguez

TRABAJO FIN DE GRADO

Para la obtención del título de

Licenciado en Matemáticas

Dirigido por

Msc. Pablo Andrés Beltrán Sosa

Realizado en el departamento de

Matemáticas

2023

Agradecimientos

"No pienses que lo que se dificulta es humanamente imposible,

si es humanamente posible está a tu alcance." (Marco Aurelio)

En primer lugar le agradezco a mi familia que siempre me ha brindado su apoyo para poder cumplir todos mis objetivos personales y académicos. Le agradezco muy profundamente a mi tutor por su dedicación y paciencia, por darnos la oportunidad de explorar otras áreas de las matemáticas, gracias por su guía y todos sus consejos. También agradecer a la universidad que me ha exigido tanto, pero al mismo tiempo me ha permitido obtener experiencias personales académicas satisfactorias y enrriquecedoras. Por último agradezco a Massimo Pigliucci el cual con sus reflexiones me dió la fuerza para siempre levantarme en los momentos mas dificiles durante mi presencia en la carrera.

Kevin Alberto Montañez Márquez

Principalmente quiero agradecerle a Dios, fuente de sabiduría y guía, por brindarme las virtudes necesarias para el desarrollo y culminación de este trabajo de grado. Al profesor Pablo por su constante apoyo y orientación, quien a través de sus amplios conocimientos, su comprensión y empatía, hizo posible la realización de este trabajo. A mi familia y abuelos que con su amor incondicional, apoyo y oraciones constantes me dieron fortaleza para no desistir. A mis amigos Pedro, Miguel, Cristian, Andrey, Duvan, Jonathan y Alan, por motivarme y creer tanto en mi y a Kevin por su paciencia, resiliencia y positivismo en todo este proceso, sin ellos nada de esto hubiera sido posible.

Juan Esteban Muñoz Rodríguez

Resumen

El Problema del Viajante (TSP) es un problema ampliamente estudiado en optimización y ciencias de la computación. Consiste en encontrar la ruta más corta para visitar todas las ciudades exactamente una vez y regresar al punto de partida.

En este trabajo se exploran los aspectos generales del problema examinando sus bases teóricas, se analizan algunos algoritmos exactos, heurísticos y metaheurísticos previamente formulados. Finalmente, se presentan tres algoritmos diseñados para solucionar el TSP y se describen y analizan los resultados obtenidos con el objetivo de enriquecer la comprensión y aplicación de estas técnicas en problemas reales de optimización.

Palabras clave: Problema del viajante, optimización, algorimos heurísticos y metaheuristicos, eficiencia.

Abstract

The Traveling Salesman Problem (TSP) is a widely studied problem in optimization and computer science. It involves finding the shortest route to visit all cities exactly once and return to the starting point.

In this work, we explore the general aspects of the problem by examining its theoretical foundations. We analyze several exact, heuristic, and metaheuristic algorithms that have been previously formulated. Finally, we present three algorithms designed to solve the TSP, describing and analyzing the results obtained with the aim of enhancing the understanding and application of these techniques in real optimization problems.

Keywords: Traveling Salesman Problem, optimization, heuristic and metaheuristic algorithms, efficiency.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

El Problema del Viajante (de ahora en adelante TSP por sus siglas en inglés) es uno de los problemas más desafiantes y estudiados en el campo de la optimización y las ciencias de la computación. Este problema, ha sido objeto de análisis y estudios exhaustivos durante décadas y trata de responder a una pregunta simple en apariencia, pero profundamente compleja: dado un conjunto de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta que permite visitar cada ciudad exactamente una vez y regresar al punto de partida?

A lo largo del presente trabajo se van a examinar las bases teóricas del TSP, revisando algunos algoritmos heurísticos y metaheurísticos previamente formulados que lo resuelven, prensentando al final tres algoritmos diseñados para solucionarlo en diferentes instancias y analizando y comparando los resultados obtenidos. Con esto, se espera contribuir al entendimiento y aplicación práctica de estas poderosas técnicas en la resolución de problemas de optimización real.

En este sentido, la estructura del documento se dividirá en ocho capítulos; los dos primeros establecen los aspectos generales del estudio, incluyendo la justificación y los objetivos del trabajo. El tercer capítulo proporciona una descripción más profunda y detallada del TSP, explora y analiza los algoritmos metaheurísticos seleccionados, aborda la Programación Matemática como tal y enuncia la formulación matemática del problema a través de la Teoría de Grafos y la Programación Matemática.

En el cuarto capítulo se presenta una visión general de cómo abordar el TSP de manera convencional sin la implementación de ningún programa computacional. Para ello, se examinan diferentes grafos, para los cuales se describe el procedimiento implementado para solucionar cada uno, junto con una explicación paso a paso de los tres algoritmos propuestos para resolver el TSP en diferentes instancias.

Los resultados y análisis se exponen en el quinto capítulo, mostrando inicialmente la eficiencia de los algoritmos en las diferentes instancias del problema y después comparando los resultados obtenidos del algoritmo propuesto y otros dos algoritmos más.

Finalmente, las conclusiones, bibliografía y anexos aparecen en los capítulos seis, siete y ocho, respectivamente.

2. Justificación

Una de las cualidades más notables y relevantes que ha permitido a los seres humanos desempeñar un papel importante en la historia del mundo, es la capacidad que poseen con respecto a otras especies de solucionar problemas complejos y desafiantes. Esta habilidad ha sido fundamental para el progreso de la humanidad, pues ha permitido establecer una gran variedad de metodologías y estrategias innovadoras mediante las cuales es posible identificar y encontrar soluciones prácticas y efectivas. De hecho, constantemente se están buscando soluciones para cada problema que aparece, aunque no necesariamente se tenga la capacidad de encontrarlas [26].

Parte de lo anterior encuentra su fundamento en la Programación Matemática (conocida también como Teoría de Optimización) un campo fundamental en matemáticas y ciencias de la computación que, en esencia, se centra en encontrar la mejor solución posible a problemas donde hay múltiples opciones, sin la necesidad de especificarlas y evaluarlas explícitamente a todas. Sin embargo, existen una gran cantidad de problemas de optimización que no pueden ser resueltos tan fácilmente debido a que se origina un fenómeno denominado como **explosión combinatorial**. Esto implica que a medida que aumenta el número de parámetros de elección dentro de un problema, el número de posibles decisiones viables y la carga computacional crecerán de manera exponencial [15]

Un ejemplo que ilustra lo anterior es el ya mencionado TSP, un problema de optimización clásico y sencillo de formular, pero lo suficientemente complicado de resolver, incluso hoy en día que se cuenta con un desarrollo tecnológico y computacional bastante avanzado. Este problema consiste en que dado un conjunto de ciudades y un "costo" entre ellas, un vendedor deberá recorrer un camino de costo mínimo, partiendo de una ciudad específica y visitar cada ciudad exactamente una vez y regresar a donde comenzó. Dicho costo puede representarse por la distancia entre las ciudades o cualquier otra medida [16].

La importancia de investigar este problema radica en su intrincada naturaleza, ya que, a pesar de ser un problema difícil de resolver, presenta numerosas aplicaciones prácticas en diferentes áreas que van desde la planificación de rutas y la logística, los mercados financieros y el turismo, hasta el diseño de circuitos electrónicos y la secuenciación de ADN, entre otros [20]. Esto ha motivado a que diversos investigadores propongan múltiples enfoques creativos a lo largo de los años para abordar el TSP, los

cuales, a pesar de no proporcionar una solución global para el problema, si ofrecen una solución local lo suficientemente cercana. La mayoría de estos enfoques suelen basarse en técnicas metaheurísticas.

Las metaheurísticas son métodos que se caracterizan por su abstracción y versatilidad en la resolución de problemas que carecen de una solución específica (como el TSP). En palabras de [14] se trata de procesos iterativos aproximados de propósito general que dirigen una heurística subyacente que ayuda a mejorar la búsqueda de soluciones en un espacio de búsqueda más amplio o complejo, combinando de manera inteligente varios conceptos con el fin de explorarlo eficazmente. Estos enfoques de optimización, generalmente inspirados en procesos naturales [19], son capaces de proporcionar soluciones adecuadas en tiempos razonables, manteniendo altos estándares de rendimiento y utilizando eficazmente los recursos disponibles, a pesar de que no proporcionan una solución global para el problema, sino soluciones locales lo suficientemente cercanas a la óptima.

3. Objetivos

Objetivo General:

- Desarrollar y aplicar un algoritmo de enjambre para resolver el Problema del Viajante (TSP), investigando y analizando en detalle diferentes métodos heurísticos y metaheurísticos que lo resuelvan para finalmente aportar, en lo posible, un enfoque novedoso que no se haya implementado antes.

Objetivos específicos:

- Utilizar la programación como medio para interpretar el problema del viajante y con ello, realizar una propuesta que lo solucione partiendo de la hipótesis de que podría existir una solución alterna que aún no se ha trabajado.
- Desarrollar un trabajo centrado en la optimización, qué es, para qué se requiere y cómo se aplica en el Problema del Viajante.
- Determinar qué procesos de la actividad matemática están inmersos y se vinculan con la programación matemática en la educación básica y media.

4. Marco Teórico

4.1. Estado del arte

Dentro de la literatura existen diversas definiciones para el TSP, una de estas es la formulada por [6] e indica que: Ün vendedor ambulante desea ir a un cierto número de destinos para vender objetos. Quiere viajar a cada destino exactamente una vez y regresar a casa tomando la ruta total más corta". El desafío consiste en hallar este recorrido evitando repetir cualquiera de las rutas transitadas.

Como ya se ha mencionado, aunque el TSP es sencillo en apariencia, resulta ser uno de los problemas más complejos de resolver debido al considerable crecimiento del número de soluciones posibles conforme aumenta la cantidad de ciudades. Lo anterior hace que el problema haga parte de una clase de problemas denominados NP-duros, lo que significa que no se conoce ningún algoritmo que resuelva este problema en un tiempo polinomial y con pocos recursos computacionales [22]. Esto hace que se vuelva necesaria la aplicación de otros métodos o técnicas para obtener, en lo posible, soluciones idóneas de alta calidad.

El origen del TSP no es del todo claro. Históricamente, los matemáticos Sir William Rowan Hamilton y Thomas Penyngton Kirman son considerados como los precursores del TSP al proponer indirectamente una definición alrededor de 1800's al considerar los denominados caminos y ciclos hamiltonianos después de crear el juego llamado **icosian game**, un tablero de madera con veinte agujeros que requería que cada vértice fuera visitado una sola vez, sin repetir ninguna arista y en donde el vértice final coincidiera con el de partida [25]. Así, Hamilton y Kirkman pudieron ser los primeros en considerar una formulación matemática indirecta para el TSP.

No obstante, se estima que el primer indicio de este problema se formuló por primera vez en 1832, en Alemania, en un libro titulado *"El vendedor ambulante: cómo debe ser y qué debe hacer para obtener comisiones y éxito en su negocio"*. En este libro se llegaba a la esencia del TSP y se indicaba que con la elección adecuada de una ruta era posible ahorrar bastante tiempo y recursos, además de dar la sugerencia de visitar cada ciudad solamente una vez [2]. Si bien se tiene constancia de que el libro fue publicado en 1832, los orígenes del TSP, matemáticamente hablando, se establecen a principios de

1930, fecha en la que se cree, comienza a aparecer dentro de la comunidad científica para ser estudiado de manera general por matemáticos de Viena y Hardvard, además de popularizarse el nombre con que se conoce actualmente.

Se dice que dentro de la comunidad científica Merril Flood fue el máximo responsable de difundir el TSP alrededor de 1930, luego de conocerlo por parte de A.W. Tucker mientras examinaba un problema de enrutamiento para un autobús escolar en New Jersey, empero, A.W Tucker argumenta ya haberlo escuchado de Hassler Whitney en la Universidad de Princeton entre 1930-31 [17]. Esto convertiría a Whitney en el pionero del TSP dentro de la comunidad científica.

Mientras tanto, al rededor de 1931 el matemático Karl Menger comienza a entrar en los detalles matemáticos para el TSP, formulándolo como "El Problema del Mensajero", que buscaba la ruta más corta que uniera una serie de puntos con una distancia entre cada par de ellos [2]. Para Menger, el TSP podía ser resuelto en un número finito de pasos, solamente que se desconocían las reglas o métodos que permitiesen reducir esta cantidad por debajo del número de permutaciones existentes, por eso llegó a considerar varios modelos y métodos para resolver el TSP: desde el clásico algoritmo de fuerza bruta hasta el planteamiento de lo que se llamaría más adelante "algoritmo del vecino más próximo", dándose cuenta de la no optimalidad de estos métodos [22].

Finalmente, durante los años 1950 y 1960, diferentes matemáticos comenzaron a interesarse en la solución de este problema esperando encontrar cada vez soluciones más exactas. Entre estas investigaciones realizadas se encuentra la contribución de los matemáticos George Dantzig, Delbert Ray Fulkerson y Selmer M. Johnson de la Corporación RAND quienes, en 1954, formularon el problema como uno de Programación Lineal en Enteros. Para solucionarlo, desarrollaron el método de los planos de corte y lo aplicaron para 49 ciudades mediante la construcción de un recorrido probando que no había uno que pudiera ser más corto [2].

Esto último se convirtió en una de las razones para la popularización del problema paralelo a su gran conexión con temas emergentes y la cantidad de aplicaciones en los problemas de combinatoria que estaban apareciendo desde la programación lineal relacionados con *la asignación* [17]. La estructura del TSP era similar a esos otros problemas, pero más difícil de resolver. Esto lo convirtió en un problema abierto para matemáticos, científicos de la computación, químicos, físicos, entre otros.

4.2. Contexto Matemático

Una vez que se tiene una idea general de lo que es el TSP y su propósito, antes de desarrollar algún método que lo resuelva es preciso formularlo de manera matemática para entender su funcionamiento. No obstante, dentro de la bibliografía consultada para la realización de este trabajo se encontraron una gran cantidad de formulaciones de las cuales, cada una cuenta con un método de desarrollo diferente. En el siguiente capítulo se mostrarán las dos formas más utilizadas para enunciarlo. La primera está asociada con la Teoría de Grafos y la segunda con la Programación Matemática.

4.2.1. TSP en Teoría de Grafos:

El TSP puede plantearse como un problema en el que según [20]:

> Dado un conjunto de n ciudades, de las cuales se conoce para cada par de ellas, la distancia que las separa, un agente viajero ha de partir de una ciudad de origen y debe visitar exactamente una vez cada ciudad del conjunto, y retornar al punto de partida. Un recorrido con estas características es llamado, dentro de este contexto, ciclo hamiltoniano o tour (p.22).

Sin embargo, antes de abordarlo de esta manera es necesario considerar algunas definiciones:

- **Grafo:** conjunto de nodos o vértices unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto.Por ejemplo, la figura 4.1 muestra la representación de un grafo cuyo conjunto de vértices y aristas están dados por los conjuntos $V = A, B, C, D, E$ y $A = (AB), (AC), (BD), (BC), (CE), (ED)$, respectivamente.

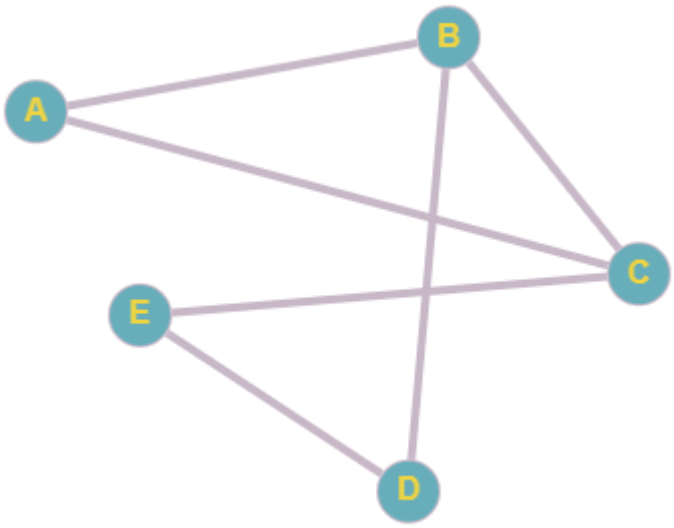


Figura 4.1: Ejemplo de grafo

- **Grafo dirigido:** grafo en el cual las aristas tienen un sentido definido. En la figura 4.2 se puede observar que, por ejemplo, las aristas que tiene como origen el vértice D tienen un sentido definido hacia los vértices B y A.

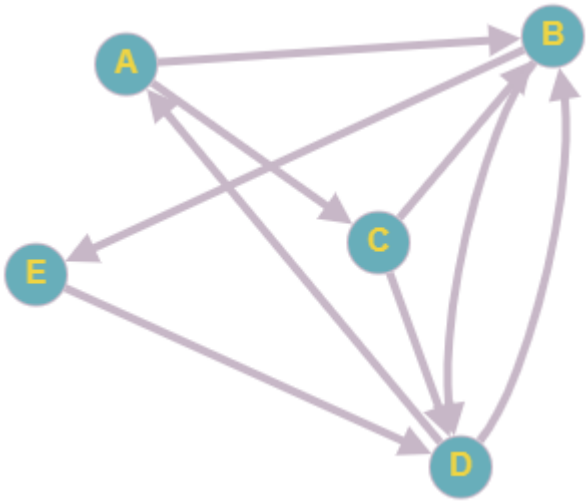


Figura 4.2: Grafo dirigido

- **Grafo completo**: grafo no dirigido que tiene entre cualquier par de nodos al menos una arista que los une. Por ejemplo, en el grafo de la figura 4.3 se puede evidenciar que ninguna arista tiene algún sentido y que además, todos los vértices están conectados entre sí.

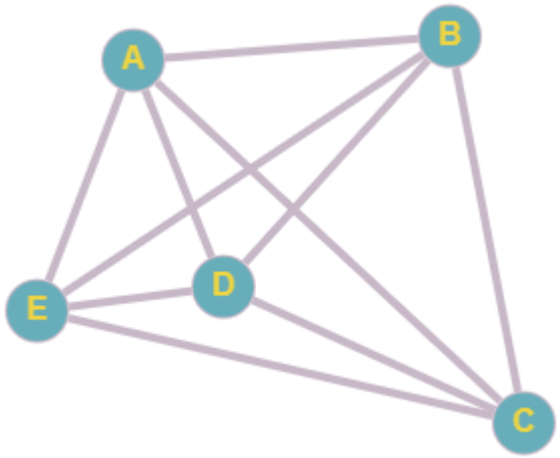


Figura 4.3: Grafo completo

- **Camino hamiltoniano:** camino que recorre todos los nodos de un grafo una y sólo una vez.

- **Ciclo hamiltoniano:** es un camino que visita todos los vértices del grafo una sola vez y regresa al vértice de partida.

Con estas definiciones, el TSP consiste en un grafo $G = (N, A, C)$ donde dado un conjunto de nodos $N = 1, ...n$, un conjunto $A = (i, j)$ de aristas y C_{ij} la matriz de distancias, se debe hallar el ciclo hamiltoniano de menor valor. Aquí, N representa a las ciudades por visitar y las aristas (i, j) la distancia entre cada una a las cuales se les asocia siempre un valor positivo de C_{ij}. Como en ocasiones no todos los nodos del grafo van a estar conectados, aquellos sin conexión serán establecidos como el mínimo de las sumas de las distancias en los caminos más cortos que unen los nodos. Adicionalmente, en este problema, no se permite permanecer en una misma ciudad durante más de una iteración [2] por lo que la distancia recorrida de un punto a sí mismo será nula, es decir, $C_{ii} = 0$.

Como un grafo siempre va a estar relacionado con una matriz de adyacencia que representa las distancias, si G es un grafo dirigido, es decir, $C_{ij} \neq C_{ji}$ entonces la matriz es asimétrica y el problema se conocerá como TSP asimétrico (ATSP). Por el contrario, si G es un grafo no dirigido, esto es, $C_{ij} = C_{ji} \forall (i, j) \in A$, la matriz de distancias es simétrica y el problema recibe el nombre de TSP simétrico (STSP) [22].

Por ejemplo, en el siguiente grafo se muestran unos nodos junto con sus distancias representadas por las aristas que los unen, cada una con su respectivo valor:

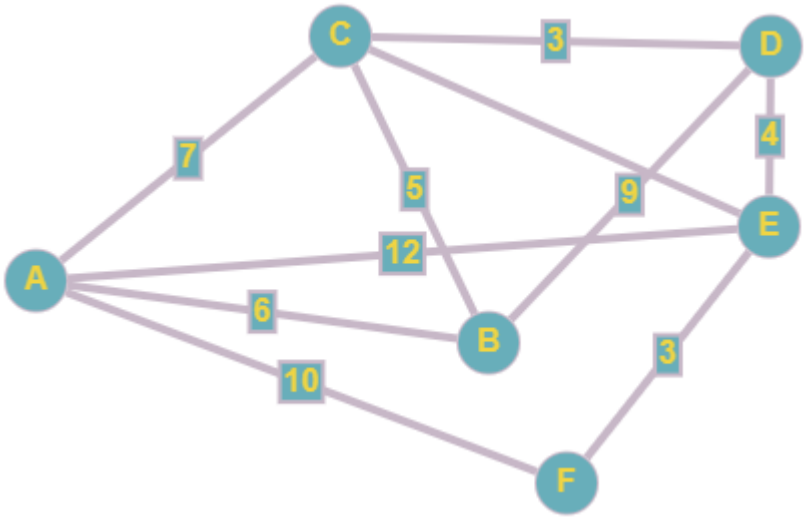


Figura 4.4: Ejemplo de ciudades representadas en un grafo

Y la matriz de distancias asociada sería:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 6 & 7 & 0 & 12 & 10 \\ 6 & 0 & 5 & 9 & 0 & 0 \\ 7 & 5 & 0 & 3 & 1 & 0 \\ 0 & 9 & 3 & 0 & 4 & 0 \\ 12 & 0 & 1 & 4 & 0 & 3 \\ 10 & 0 & 0 & 0 & 3 & 0 \end{pmatrix} \end{array}$$

4.2.2. TSP en Programación Matemática

Existen diferentes formulaciones desde la Programación matemática para el TSP. La siguiente se cataloga formalmente como una de las más aceptadas por la comunidad científica [13] y surge de un estudio realizado en 1954 por los matemáticos Dantzig, Fulkerson y Johnson. Esta consiste en:

Dado un conjunto de n ciudades $N = 1, 2, 3, ..., n$ y un conjunto de caminos $(i, j) \in A$ que une cada una de las ciudades. Si C_{ij} es la distancia requerida para ir de la ciudad i a la ciudad j, donde $C_{ij} = C_{ji}$ y la variable de decisión del problema X_{ij} es:

$$X_{ij} = \begin{cases} 1, \ si\ el\ camino\ va\ de\ i\ a\ j \\ \\ 0, \ en\ otro\ caso \end{cases}$$

Entonces, el problema asume el siguiente modelo matemático:

$$minZ(x) = \sum_{(i,j)\in A} C_{ij} \cdot X_{ij} \tag{1}$$

Sujeto a:

$$\sum_{i:(i,j)\in A} X_{ij} = 1 \quad \forall j \in V \tag{2}$$

$$\sum_{j:(i,j)\in A} X_{ij} = 1 \quad \forall i \in V \tag{3}$$

$$\sum_{(i,j)\in A: i\in U, j\in (V-U)} X_{ij} = 1 \quad 2 \leq |U| \leq |V| - 2 \tag{4}$$

Donde A corresponde al espacio de búsqueda y C_{ij} a la distancia asociada a la variable de decisión X_{ij}. La ecuación (1) es la función objetivo y minimiza el camino del viajante. La ecuación (2), señala que a cada ciudad j solo se puede llegar desde una ciudad i. La ecuación (3) es una restricción que indica que desde una ciudad i solo se puede salir por un único camino. Finalmente, la ecuación (4) impide que se generen subtours dentro del grafo.

4.2.3. Métodos de resolución para el TSP

En general, se puede asegurar que el TSP tiene una única solución, pues en teoría, siempre es posible evaluar todas las soluciones y de ahí seleccionar la que representa un menor coste ya sea en recorridos, precios, tiempo, entre otros.El inconveniente es que la cantidad de posibles soluciones va a aumentar de manera significativa cuando se incremeta las instancias del problema y como se ha venido mencionado, lo ideal para el TSP sería encontrar la ruta más óptima en una cantidad n de ciudades.

Lo ideal sería pensar que el TSP siempre puede ser resuelto, sin embargo, en [6] se indica que en un grafo de tres o más nodos el número máximo de ciclos hamiltonianos está dado por: $\frac{(n-1)!}{2}$, donde

n es la cantidad de nodos o ciudades consideradas dentro del problema, algo que, en teoría, puede ser analizado uno a uno por una computadora. El problema es que a medida que n aumenta, $\frac{(n-1)!}{2}$ se vuelve increíblemente grande provocando que cualquier algoritmo que se utilice para encontrar la solución óptima experimente un incremento desmedido en el tiempo de cálculo, generando errores por sobrecargas en la memoria RAM del ordenador. En la tabla 4.1 se expone lo anteriormente mencionado:

n	$\frac{(n-1)!}{2}$
3	1
4	3
5	12
6	60
7	360
8	2520
9	20160
10	181440
25	$3{,}10224201 \times 10^{23}$
30	$4,420881 \times 10^{30}$
45	$3,0207631552 \times 10^{52}$

Cuadro 4.1: Cantidad de ciclos Hamiltonianos en un Grafo

Por esta razón, aunque el problema puede resolverse en una cantidad finita de pasos, esto no es suficiente por lo que se requiere desarrollar otras estrategias que reduzcan, por lo menos, el tiempo y el esfuerzo de cálculo. Hasta el momento no se ha encontrado ninguna que dé una solución general para cantidades superiores a 50 ciudades, pese a esto, sí se han hallado algunos algoritmos para casos particulares del problema que, si bien a veces no garantizan una solución óptima, arrojan soluciones lo bastante cercanas, esto depende de la capacidad de cómputo y los recursos disponibles.

En lo que sigue, se describirán con detalle algunos de los algoritmos que han sido utilizados para encontrar soluciones para el TSP, distinguiendo tres categorías: algoritmos exactos, algorítmos heurísticos y algoritmos metaheurísticos.

4.2.4. Algoritmos exactos

Los algoritmos exactos, a pesar de asegurar una solución óptima para el TSP, no se recomiendan para instancias de tamaño mediano o grande, ya que estos consisten en formular, evaluar y comparar todas las posibles permutaciones, siendo el número de soluciones posibles 4.2.3. Esto hace que sea inabordable tan pronto aumenta el número de parámetros por lo cual todavía no se ha podido garantizar que algún algoritmo exácto pueda ejecutarse en un tiempo razonable.

En [2] se menciona que a lo largo de la historia, además de los algoritmos expuestos, se han planteado muchos otros métodos como el de los planos de corte (Cutting planes), de ramificación y acotación (Branch and Bound) o de ramificación y corte (Branch and cut). Estos algoritmos comienzan con la búsqueda de la solución del problema relajado, esto es, resolver el problema sin contemplar el conjunto de restricciones que "fuerzan la integrabilidad de las variables"[11], para luego añadir aquellas limitaciones que no se necesiten hasta que se obtenga la solución entera.

A continuación, se muestran algunos de los más conocidos:

Algoritmo de Dijkstra

Se ha demostrado que el algorimto de Dijkstra es uno de los algoritmo exactos que presenta un buen desempeño en la mayoría de los casos al buscar la mejor solución, aun así, cuando los grafos son demasiado grandes el algoritmo suele requerir de un gran tiempo de procesamiento para encontrar la respuesta [24]. De cualquier manera, es uno de los algoritmos más simples para hallar la ruta de caminos mínimos en un grafo G desde un nodo S a todos los nodos alcanzables desde S y se utiliza cuando el grafo no tiene aristas negativas y es dirigido.

El algoritmo consta de un etiquetado particular $[\eta, B]_{(\alpha)}$. Aquí, η representa la distancia acumulada que hay desde el inicio hasta el nodo, B al nodo predecesor que se agrega al nodo anterior que se está etiquetando y α representa la iteración. El algoritmo genera un conjunto con todos los nodos del grafo, del cual se escoge el que tenga la etiqueta con menor valor. Este nodo queda marcado permanentemente y es retirado del conjunto inicial y adicionado al conjunto de los nodos que quedan marcados.

A continuación, se actualizan las etiquetas de los sucesores del nodo seleccionado de modo que la distancia del nodo sucesor tenga la menor distancia acumulada y equivalga al valor de la etiqueta del nodo seleccionado más el costo. Este proceso se repite hasta observar todos los nodos adyacentes al inicial para después seleccionarlo y marcarlo permanentemene. En palabras de [24], este algoritmo se basa principalmente en marcar los nodos de una forma especial: el nodo que se marcará es seleccionado entre todos los nodos disponibles, la única condición es que debe tener la menor distancia acumulada en el camino desde el nodo de origen.

En el caso en que hayan varios nodos con la misma distancia se puede seleccionar cualquiera de ellos aleatoriamente, dado que más adelante serán seleccionados y marcados. Esto garantiza que una vez

que todos los nodos se hallan marcado, se encuentre además la distancia mínima.

Algortimo Bellman-Ford

El algortimo de Bellman-Ford es, en estructura, similar al algoritmo de Dijkstra con la diferencia de que en este se permiten entradas de aristas con pesos negativos y se considera todas las aristas $|V-1|$ veces siendo $|V|$ la cantidad de vértices o nodos [21].

La estructura general del algoritmo brindada por Mukhlif y Saif,(s.f., como se citó en [27]), es: dado un grafo G con n nodos, el número máximo de aristas que puede haber en un camino más corto es igual a $n-1$. Si el camino más corto tiene más de $n-1$ aristas; análogamente, si un grafo contiene un ciclo negativo, inevitablemente tendrá un camino más corto con n o más aristas.

El algoritmo de Bellman-Ford determina las rutas más cortas de forma ascendente, calculando inicialmente las distancias mínimas que tienen como máximo un borde en la trayectoria. Posteriormente, calcula los caminos más cortos con 2 aristas como máximo, y así sucesivamente. Cuando llega a la i-ésima iteración, se calculan los caminos más cortos con máximo i aristas. La idea es que, asumiendo que no hay un ciclo de peso negativo, si se han calculado las rutas más cortas con un máximo de i aristas, entonces una iteración sobre todas las aristas garantiza que se obtenga la ruta más corta con un máximo de $(i+1)$ aristas (GeeksforGeeks, 2012, como se citó en [27]).

4.2.5. Algoritmos heurísticos

Para hablar de los algoritmos heurísticos es preciso mencionar el origen etimológico de la palabra heurística dado que de allí proviene la relación de aprendizaje que se busca imitar al realizar la programación. La palabra heurística proviene del griego *heuriskein* y significa *hallar* o *inventar*. Esta palabra hace referencia a la capacidad de solucionar problemas o generar estrategias a través de la experiencia en el ámbito de la programación buscando solucionar situaciones mediante la creatividad o por medio del ensayo y error.

En palabras de [22]: "Los algoritmos heurísticos son un conjunto de técnicas que permiten resolver los problemas de manera eficiente, aunque no se puede asegurar que la solución obtenida resulte ser la solución óptima del problema"(p.19). Estos algoritmos son de buena calidad y ayudan a encontrar soluciones bastante eficientes en tiempos aceptables usando esfuerzos computacionales relativamente

bajos contrario a los algoritmos exáctos que proporcionan la solución óptima del problema. Cabe mencionar que las heurísticas son algoritmos que dependen del problema, en otras palabras, se diseñan para problemas específicos y su uso solo tiene sentido para dichos problemas [19].

Una posible clasificación para estos métodos podría ser la proporcionada por [11] y es:

- *Algoritmos constructivos*: heurísticos que tratan de construir una solución factible sin precisar de ninguna previa.
- *Heurísticas de multiarranque*: trabaja con probabilidades y procesos aleatorios que otorgan mejores soluciones.
- *Heurísticas voraces*: hace uso del mejor elemento, teniendo en cuenta el objetivo a optimizar, pero sin tener en cuenta consecuencias futuras, ya que tras tomar una decisión esta no vuelve a ser considerada en futuras iteraciones.
- *Heurísticas de mejora*: la idea principal es comenzar con una solución inicial a la que se le quitarán k aristas para después añadir la misma cantidad mejorando la solución.

Algoritmo del vecino más cercano

El algorimto del vecino más cercano es una heurística de mejora de las más sencillas e intuitivas para resolver el TSP. Este algoritmo trata de construir un ciclo hamiltoniano de menor distancia buscando la ciudad más próxima a un nodo dado y añadiéndola al recorrido. Esto se repite hasta que se recorran todos los nodos para que cuando se pase por todos, se regrese a la ciudad inicial. Parte de su formulación actual se debe a Rosenkrantz, Stearns y Lewis (1977, como se citó en [20]), sin embargo, la primer definición para este algoritmo fue dada por Karl Menger en 1931.

En este algoritmo es frecuente iniciar bien conforme se construyen las soluciones para el problema y se realizan las trayectorias de las ciudades más cercanas con una distancia corta, sin embargo, a medida que el algoritmo sigue la solución suministrada comienza a empeorar, especialmente al final. La razón es que constantemente pueden aparecer distancias aún más largas entre nodo y nodo, lo que se conoce como "miopía del procedimiento"[22], pues cada vez se van escogiendo opciones mejores sin advertir problemas posteriores.

Algoritmo de Christofides

El algoritmo de Christofides o algoritmo de Christofides-Serdyukov[7], es un algoritmo aproximado desarrollado por Nicos Christofides en 1976 y es especialmente eficaz para instancias del TSP en grafos completos que satisfacen la desigualdad triangular. Este profesor del Imperial College en Lóndres, se percató de que sus resultados podían mejorarse empleando el concepto de "de árbol soporte de mínimo peso asociado a un grafo"[19], a pesar de que esto aumentara la complejidad del algoritmo.

Este algoritmo es reconocido por ser, hasta la fecha, uno de los mejores para resolver el TSP en grandes instancias y en una métrica arbitrarea, garantizando una solución que es, como máximo, 3/2 veces más larga que la solución óptima. Asimismo, para la métrica euclidea, el TSP puede ser resuelto completamente en cierta cantidad de instancias con decenas de miles de ciudades en una pequeña fracción del 1 % .

No obstante, no siempre es la mejor opción para todas las instancias del TSP, dado que su rendimiento puede variar según las características del problema, a pesar de esto, el algoritmo es ampliamente utilizado en la práctica debido a su garantía de aproximación y su capacidad para proporcionar soluciones de calidad en un tiempo razonable. Puede, al igual que el TSP, utilizarse en la planificación de rutas, logística, diseño de circuitos, entre otras.

4.2.6. Algoritmos metaheurísticos

Las heurísticas tienen ciertas ventajas para resolver situaciones de optimización en problemas combinatorios, sobretodo en velocidad y que las soluciones suelen ser bastante aceptables. Sin embargo, habitualmente estos métodos tienen un limitante que hace que no sean tan eficientes en ciertos niveles y donde no puede asegurarse que estas soluciones sean las mejores en relación a modificaciones menores a nivel local. [4]. Como resultado, es necesario realizar una mejora que refine las soluciones obtenidas por las heurísticas puesto que puede suceder que se estanquen en soluciones de baja calidad cuando se enfrenten a cambios en las instancias.

En respuesta de lo anterior, en los últimos años han surgido una serie de métodos bajo el nombre de *metaheurísticas*. Estos se centran en diseñar técnicas de propósito general para guiar la construcción o búsqueda de soluciones locales en distintas heurísticas [4]. El término fue acuñado en 1986 por Fred Glover, pero en algunos textos puede encontrarse la expresión "heurísticos modernos" para referirse al

mismo concepto (Reeves, 1995, como se citó en [20]).

Los procedimientos metaheurísticos constituyen una categoría de enfoques aproximados diseñados específicamente para abordar problemas complejos de optimización combinatoria, que escapan de la efectividad de los heurísticos tradicionales. Estos métodos ofrecen un marco amplio y versátil para la creación de algoritmos híbridos mediante la combinación de diversos conceptos provenientes de campos como la inteligencia artificial, la evolución biológica y los mecanismos estadísticos (Osman y Kelly, 1995, como se citó en [20]).

Algortimos Genéticos

Los algoritmos genéticos (AG de ahora en adelate) son algoritmos de búsqueda y aprendizaje basados en la evolución natural. Dicho proceso es una adaptación conceptual de la teoría de la evolución de las especies de Charles Darwin [19], quien declaraba el proceso como *"descendencia con modificación"*. Darwin proponía que la vida de las especies del planeta cambiaba de acuerdo al tiempo y que solo aquellos que eran capaces de adaptarse al entorno eran los que prevalecían y sobrevivían.

Estos algoritmos de optimización suelen recrear un proceso evolutivo realizando variaciones en una instancia cromosómica. Tales cromosomas están basados en lo que en biología se conocen como *Genotipos* y *Fenotipos* los cuales se alteran en distintas iteraciones para obtener una solución (sea óptima o no) en cada intento. Un algoritmo genético suele estar comprendido en dos modelos expresados por [12] como el modelo generacional y el modelo estacionario.

- **Modelo Generacional:** Durante cada iteración se crea una población completa con nuevos individuos, esta población reemplaza la anterior.

- **Modelo estacionario:** Durante cada generación se toman dos padres de la población y se les aplican operadores genéticos. En este caso el, o los descendientes, reemplazan uno o varios cromosomas de la población inicial. De igual modo [12] explica que este modelo es elitista, significando esto que en la población se tomará a los mejores padres del conjunto, estos son los que devuelven los resultados más óptimos en una instancia cualquiera en alguna iteración.

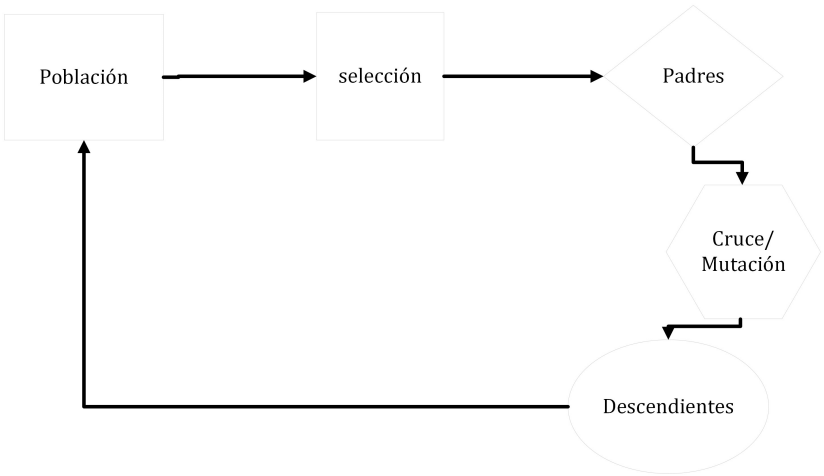


Figura 4.5: Diagrama Algoritmo Genético

El objetivo de un algoritmo genético es encontrar una solución óptima a un problema (o la solución más eficiente posible) en un tiempo razonable que maximice o minimice una función de adaptación con un conjunto de parámetros establecidos. Estos algoritmos trabajan con una población de individuos x_i, cada individuo representa un punto de búsqueda en el espacio de soluciones potenciales a un problema dado [10].

$$p(t) = \{x_1^t, ..., x_n^t\}$$

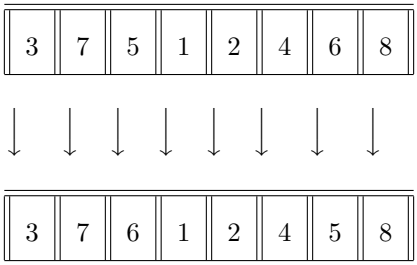
Cada uno de los individuos x_i es evaluado en una función de adaptación, la cual permite destacar a los mejores individuos del conjunto. Luego de ordenar a los individuos del mejor al peor desempeño, se toman a los de mejor *fitness* dependiendo de un criterio de selección previamente adaptado donde se les aplica algún operador de mutación o cruce para proceder a reemplazar en el algoritmo y repetir el proceso hasta obtener el resultado esperado.

El operador de mutación o cruce se aplica al **Genotipo** para obtener un **Fenotipo**. El primero hace referencia (en biología) al conjunto de genes e información genética que conforman a un individuo de cualquier especie; en cambio, el segundo es la expresión en forma física de las características de un individuo de cualquier especie. Así por ejemplo el genotipo de una persona que tiene los ojos verdes es la carga genética y el fenotipo es precisamente el verde que resulta de la configuración específica del genotipo.

La importancia de esto radica, en que para recrear un individuo en un algoritmo genético se usa

la representación de una posible solución simulando un Genotipo como mecanismo para codificarlo. De esta manera, para el problema del TSP, los individuos se representan como secuencias de permutaciones de números reales que, dependiendo de su desempeño, son tomados o no en cuenta para las siguientes iteraciones. En dicho caso el Fenotipo es un valor numérico que representa el desempeño de un Genotipo.

En un caso puntual, para una instancia de ocho ciudades para el TSP, una consecución de lugares a visitar sería una permutación que configuraría un individuo, cada una de las ciudades representa un cromosoma del mismo. Una mutación consiste en cambiar un elemento por otro para configurar la *evolución* de este.



Los algoritmos genéticos son de tipo estocástico, lo que significa que usan procesos de búsqueda probabilísticos al realizar permutas en los genotipos para después seleccionar de manera aleatoria individuos y encontrar los que mejor se adapten. Una alternativa puede ser la elección por ruleta, la cual asigna una probabilidad proporcional al valor de costo del cromosoma y luego simula una ruleta en la que los espacios tienen correspondencia con su porcentaje.

Algoritmo de Optimización de Colonia de Hormigas

Los algoritmos de optimización por colonia de hormigas o *Ant Colony Optimization* (ACO por sus siglas en inglés) son algoritmos metaheurísticos inspirados por el comportamiento estos insectos. Este concepto fue introducido por Dorigo M. en los años 90 y fue propuesto como herramienta para solucionar problemas de optimización complejos en una cantidad razonable de tiempo de cómputo [3]. También es importante destacar que los principios de autorganización que facilitan la coordinación altamente eficiente de las hormigas reales, pueden ser utilizados para coordinar grupos de agentes artificiales que trabajen en conjunto para abordar desafíos computacionales [8].

La idea conceptual del algoritmo es imitar el comportamiento natural de las hormigas, haciendo una analogía con la necesidad que tienen de llevar comida hasta su nido y cómo ello comprende un proceso de realimentación grupal que deriva en encontrar el camino más corto del nido hasta dicho alimento;

de ahí que salgan de su nido e inicien un proceso de búsqueda de comida de manera aleatoria en un espacio físico. Al encontrar el alimento, las hormigas toman una pequeña parte para llevarla al nido, desprendiendo durante su regreso una sustancia química llamada *feromona*, que servirá de guía para que otras hormigas encuentren el alimento.

Este proceso de colaboración a través del medio físico se conoce como **Estigmergia** y hace que las hormigas coordinen sus actividades mediante la exploración del entorno y la modificación del mismo cuando se mueven [8]. Esta retroalimentación y comunicación entre las hormigas provoca que después de cierto tiempo puedan encontrar las rutas más cortas entre el nido y el alimento.

Representación del funcionamiento de un ACO:

La siguiente es una representación inspirada en la de [3] recreando a su vez a [8].

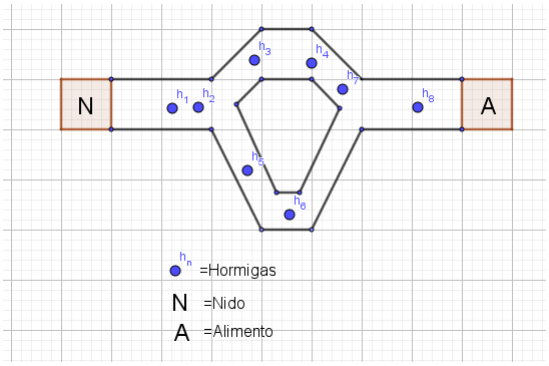


Figura 4.6: Hormigas y caminos. Elaboración propia.

La imagen 4.6 representa un camino desde el nido hasta un punto en el que se divide en dos. Para conseguir el alimento, las hormigas deben tomar alguna de las dos rutas, recoger un poco y volver al nido dejando un rastro de feromonas. Como en el momento inicial no hay ninguna feromona esparcida, las hormigas tomarán un camino de manera aleatoria siguiendo algunas reglas probabilísticas. De manera inicial, por cada uno de los caminos irán la misma cantidad de hormigas.

Suponiendo que todas las hormigas caminan a la misma velocidad, pude notarse que cruzarán más hormigas por el camino corto en el mismo periodo de tiempo que por el camino largo. Como están cruzando más hormigas por un camino que por el otro, la feromona se concentrará en dicho camino. Después de un tiempo considerable, la cantidad de feromona en el camino corto será lo suficientemente alta como para alterar la decisión del conjunto total de hormigas, que ya no tomarán el camino de forma aleatoria, sino el que designa la colonia [3], así, en algún momento todas las hormigas encontrarán el

camino óptimo. Este fenómeno lo describe [8] como *retroalimentación positiva*, que se puede traducir como la utilización del camino más corto.

Para el caso del TSP, se comienza contemplando un conjunto de m hormigas artificiales que se mueven de una ciudad a otra iniciando en una cualquiera y en donde, dependiendo de las aristas que recorran en cada iteración, se alterará el rastro de feromonas debido al movimiento realizado. Cuando todas las hormigas hayan completado el recorrido, la que logre la menor distancia agrega a las aristas que han formado parte de su ruta una cantidad que es inversamente proporcional a la longitud de dicho recorrido [19].

Como los ACO son principalmente algoritmos constructivos, cada vez que una hormiga realice un recorrido y cruce por las aristas de su elección en todas las iteraciones del proceso irán generando soluciones más cercana a la óptima. Así, cada arista del grafo representará los posibles movimientos que las hormigas pueden realizar y lo hacen guiadas de las siguientes informaciones [9]:

- **Información Heurística:** mide la preferencia n_{pq} de moverse del nodo p al q recorriendo la arista que los conecta. También se define como la visibilidad y se plantea como el inverso de la distancia del camino.

- **Información de rastros de feromonas artificiales:** miden la deseabilidad aprendida del movimiento de las hormigas T_pq de los vértices p a q, esto es lo que imita a la feromona depositada por las hormigas en la vida real, esta información va cambiando dependiendo de las soluciones encontradas por las hormigas.

Cuando todas las hormigas hayan generado una solución, se evalúa y se deposita una cantidad determinada de feromona que dependerá de la calidad de la solución. Lo anterior influirá directamente al resto de las hormigas en el futuro generando una inclinación cada vez mayor hacia algunas soluciones.

Por otro lado, el proceso de construcción de la solución óptima en un algoritmo ACO incluye dos procedimiento adicionales denominados *evaporación de feromona* y *acciones del demonio* [9]. El primero es efectuado por el entorno y se emplea para permitir que las hormigas busquen y exploren nuevas regiones del espacio de búsqueda, evitando el estancamiento en óptimos locales. Las acciones del demonio son opcionales y no tienen una correspondencia con el comportamiento de las hormigas reales y se implementan para llevar a cabo tareas desde una perspectiva global.

Construcción de una solución por la hormiga K

Como la elección de una hormiga de ir de un nodo a otro es completamente aleatoria al principio y luego se basa en la acumulación de feromonas y la distancia entre esta y la ciudad, es necesario considerar para cada hormiga una memoria $F(p_i)$ para almacenar las soluciones parciales o ciudades visitadas. Dicha memoria es reiniciada al comienzo de cada recorrido por lo que se inicia con una solución parcial vacía que irá aumentando en cada iteración, después de añadir un elemento de solución viable seleccionado probabilísticamente entre los vecinos de la solución actual [9]. Lo anterior implica encontrar el camino del grafo que establece el recorrido más corto.

De esta manera, el modelo de probabilidad seguido por una hormiga k descrito por [9] es:

$$P_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta ij^{\beta}}{\sum_{h \in F(p_i)}^{l}} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}, j \in F(p_i); \\ 0, \qquad\qquad en\ otro\ caso \end{cases}$$

En donde:

- $F(p_i)$ es el conjunto de puntos viables para el punto p_i
- τ_{ij} es el valor de feromona asociado a la arista (pi, pj).
- $\eta_{ij} = \frac{1}{d_{ij}}$ es el valor heurístico asociado a la arista (pi, pj). d_{ij} es el inverso de la distancia del camino
- α y β son parámetros positivos que determinan la importancia relativa de la feromona con respecto a la información heurística.

Dado a que en los recorridos que realicen las hormigas los niveles de feromona aumentarán o disminuirá en los caminos dependiendo qué tan buenos sean, es necesario llevar a cabo un proceso de *actualización del rastro de las feromonas*, teniendo en cuenta su evaporación. Primero se reducen todos los valores de feromona por medio del proceso de evaporación luego, se incrementa el nivel de feromona en las soluciones que fueron buenas. De esta manera, aquellos caminos que casi no fueron visitados se volverán prácticamente invisibles para las hormigas en comparación con los otros que serán transitados con mayor frecuencia.

Para realizar lo anterior, en [9] se presenta de la siguiente manera:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \Delta\tau_{ij}$$

Donde:

- $\rho \in (0, 1]$ es el factor de evaporación.
- $\Delta\tau_{ij} = \sum_{k=1}^{\#Hormigas}$, $\Delta^k_{\tau_{ij}}$ es la acumulaciónn del rastro proporcional a la calidad de las soluciones.
- $\Delta^k_{\tau ij} = \begin{cases} \frac{1}{L_k}, & \text{si la hormiga } k \text{ utlizó la arista } (p_i, p_j) \\ 0, & \text{en caso contrario} \end{cases}$
- L_k es la constante de actualización de la feromona

Algoritmo de Optimización por Enjambre de Partículas

Al igual que los algorimos genéticos, la estrategia de enjambres de partículas es una excelente alternativa para solucionar problemas de optimización. Fue propuesto por Kennedy y Eberhart, en 1995 y su intención inicial se encontraba dentro de la inteligencia artificial basándose en analogías simples de la interacción entre poblaciones. En un principio estas analogías intentaban asemejarse a las bandas de pájaros, pero con el tiempo se convirtieron en lo se denominó como *optimización por enjambre de partículas* o *Particle Swarm Optimization*, PSO por sus siglas en inglés [1].

Kennedy y Eberhart realizaron un esquema donde trataban de explicar las actividades sociales de distintos animales que se relacionan en un enjambre, como las aves o los peces. Categóricamente un enjambre puede definirse como "un grupo de individuos que se comunican directa o indirectamente entre ellos, actuando en su respectivo ambiente"([18],(p.6)).

Como tal, en el algoritmo los individuos se ven como puntos los cuales son llamados partículas. Estas se mueven a distintas velocidades y parámetros en diferentes posiciones en un espacio de búsqueda. Al principio del algoritmo, la población toma posiciones aleatorias para que las partículas se desplacen por el espacio recordando cual ha sido la mejor posición encontrada; luego una partícula le comunica a las otras la mejor posición.

Mientras pasa el tiempo, y en el transcurso de una iteración, la partícula ajusta su posición y velocidad con base a lo que las otras partículas le comuniquen, esto deriva en que el enjambre tienda a dirigirse al mejor lugar en el espacio de búsqueda minimizando el costo.

Kennedy y Eberhat describen los parámetros en el procedimiento de búsqueda en un PSO

mediante las siguientes expresiones:

$$v_i{}^{k+i} = wv_i{}^k + c_1 rand_1(pbest - x_i{}^k) + c_2 rand_2(gbest - x_i{}^k) x_i{}^{k+1} = x_i{}^k + v_i{}^{k+1}$$

Aquí:

- c_1 y c_2 son constantes definidas como coeficientes de aceleración
- w es el factor de inercia
- $rand_1$ y $rand_2$ son dos números aleatorios en el intervalo $[0, 1]$
- x_i representa la i-ésima partícula
- $pbest_i$ representa la mejor posición previa de x_i
- $gbest$ es la mejor posición de todo el enjambre
- v_i es la velocidad de la partícula x_i

Estos parámetros tienen un gran peso en el comportamiento del algoritmo, soluciones y tiempos de ejecución. Si se alteran los coeficientes de aceleración y se emplea un coeficiente cercano a cero provocará una búsqueda muy fina en una región, pero no muy extensa; al contrario de cuando se usan coeficientes cercanos a uno, no será tan específica la búsqueda, pero será mucho más amplia.

Finalmente, se propone el siguiente diagrama donde se ilustran las diferentes heurísticas y metaheurísticas más conocidas, algunas expuestas en este trabajo.

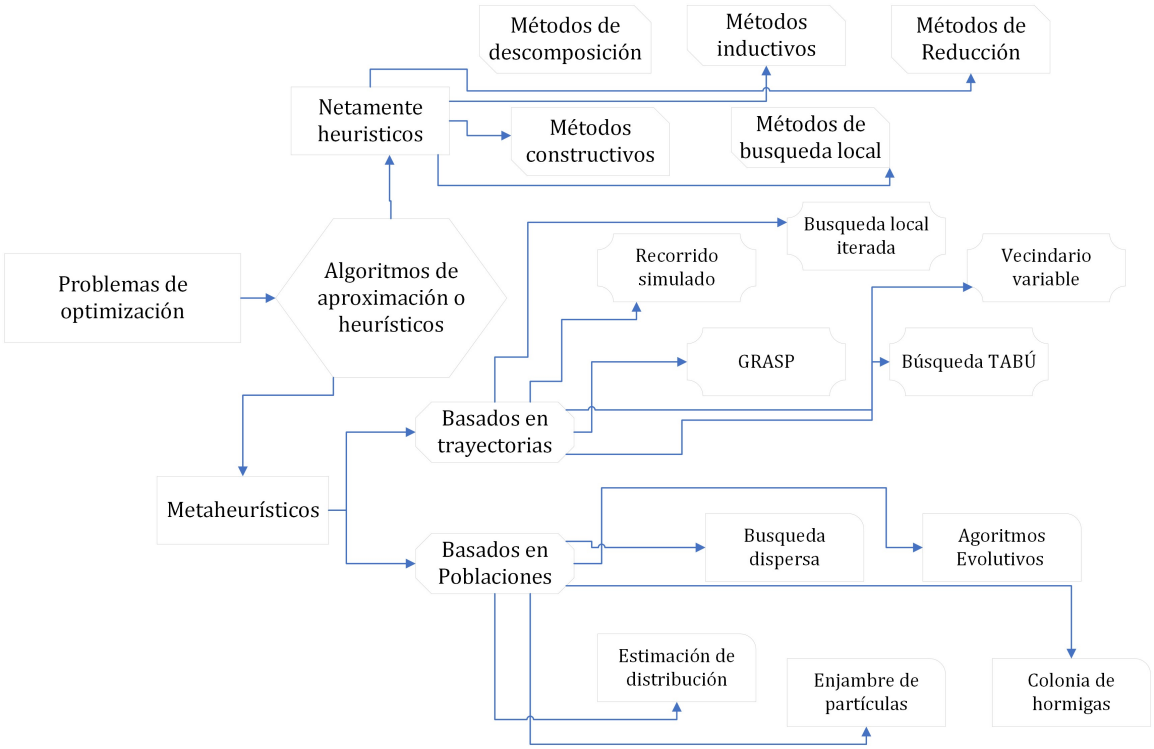


Figura 4.7: Diferentes heurísticas y metaheurísticas

4.2.7. Indagación

Para encontrar documentos académicos que soporten la teoría, se decidió realizar una Revisión Sistemática de Literatura entorno a investigaciones que pudieran dar cuenta de intentos de solución para el TSP solamente por medio del algoritmo de optimización por colonia de hormigas (ACO) y el algoritmo genético (AG). Este método surge de la idea de adaptar la búsqueda que se realizó en el marco de las investigaciones de la Universidad del Cauca y la Corporación Universitaria Comfacauca por parte de [23].

Los autores afirman que normalmente se suelen presentar soluciones al problema a través de algoritmos donde se recrean los comportamientos de las hormigas de forma muy general. Por lo tanto, se plantearon si era posible crear una variante más eficiente luego de incluir características específicas que normalmente no se tienen en cuenta al trabajar con un algoritmo de hormigas (arrieras o cultivadoras de hongos en este caso).

En consecuencia, no solo era necesario encontrar soporte académico que mostrara los antecedentes para resolver el TSP por medio de lo ya dicho, sino que también tenía que ser lo suficientemente específico para encontrar si ya alguien había trabajado con dicha variante u otras variantes de los algo-

ritmos. De modo que, con base en su investigación, se decidió hacer una indagación mediante cadenas de búsqueda con conectores lógicos (o booleanos) en distintas bases de datos (Scielo, Dialnet, Redalyc) y Google académico aprovechando las palabras clave y los objetivos del trabajo.

Cadenas de Búsqueda utilizada:

(.ºptimización por colonia de hormigas.ºR.ªCO.ºR.ºCH") AND("TSP.ºR"Problema del viajante.ºR"problema del vendedor viajante") +"variante-"tipos"

(.ªlgoritmos genéticos.ºR.ªG") AND("TSP.ºR"Problema del viajante.ºR"problema del vendedor viajante") +"variante-"tipos"

Estas cadenas permiten realizar búsquedas muy específicas, lo que deriva en encontrar solo unas decenas de documentos académicos los cuales son potencialmente útiles para la indagación. Así mismo, se tuvo en cuenta un proceso de revisión aplicando principios de inclusión y exclusión de documentos, como la coincidencia de los títulos de los archivos, resumen, palabras claves, conclusiones, archivos repetidos y relevancia (cantidad de citas).

4.3. Solución del TSP con Algoritmos de Optimización de Hormigas o Genéticos

Ruteo dinámico

El estudiante Sergio Román Cherchyk del Instituto Universitario Aeronáutico con ayuda de la ingeniera Daniela Díaz (2016), utilizaron el algoritmo de colonia de hormigas para encontrar las rutas aéreas mas eficientes para solucionar el problema del TSP utilizando las 23 capitales de las provincias de Argentina representadas en un grafo.

En este estudio se replican los comportamientos base de un ACO, como la preferencia de caminos con altos niveles de feromona, mayor crecimiento de feromonas por caminos cortos y comunicación entre hormigas por un rastro, pero en este caso se adaptó el algoritmo con la intención de darle a las hormigas otras capacidades no naturales para complementar el algoritmo, como memoria de trabajo, la cual les permite recordar las ciudades ya visitadas y la capacidad de determinar distancias entre ciudades.

A continuación, se utilizó un método propuesto por Dorigo llamado *Ant Cycle* en el cual la actualización de feromona se da después de haberse completado todos los recorridos, a comparación del

método más tradicional (o más primitivo como se expresa en el trabajo) en el cual la actualización de feromona se realiza luego de que cada hormiga pasa por cada arco.

En ese mismo orden de ideas, es interesante como por prueba y error los investigadores cambiaron los valores de la feromona, ya que la importancia del parámetro radica en la calidad de soluciones que el algoritmo devuelva, si los valores de incidencia de la feromona son muy bajos ocasionará que las hormigas se concentren en algún mínimo local, estancando el programa y brindando soluciones pobres. Al contrario, si el valor de las feromonas es alto, la evaporación de esta estará muy reducida, siendo necesario que pasen demasiadas iteraciones para revelar patrones que se diferencien y que puedan influir en las decisiones de las hormigas.

Modelo híbrido, ACO y AG

Gustavo García y Andrés Ramírez (2019) utilizaron un ACO para solucionar un problema de ruteo de vehículos para la ciudad de Pereira, ellos realizaron un híbrido en el cual combinaron el algoritmo de Optimización de Colonia de Hormigas con un algoritmo Genético, mas específicamente el algoritmo de Chu-Beasley.

El algoritmo Chu-beasley es una versión modificada del algoritmo genético básico, el cual se centra en mantener diversidad en los cromosomas que conforman la población en el desarrollo del mismo. Su objetivo es cambiar solo un cromosoma en los individuos de la población buscando con ello que en cada generación se puedan encontrar soluciones de alta calidad y garantiza la diversidad de la población durante todas las siguientes generaciones.

Para la creación del algoritmo se tuvieron en cuenta parámetros como la cantidad de camiones, la distancia promedio que recorrían, la hora de salida y llegada de estos; también se tomaron como vértices del grafo 32 hospitales y clínicas de la ciudad. En seguida se calcularon las distancias mediante un algoritmo en Python que utiliza una conexión a la aplicación de *Google Maps* para establecer la longitud entre los nodos.

Luego de calcular las distancias, se entrega dicha información a la colonia de hormigas la cual construye rutas. Comienzan estas con el proceso de recorrer el grafo y depositar feromonas. La feromona se actualizará una vez todas las hormigas hayan añadido un nodo. Terminado el proceso de construcción de rutas se envían al AG Chu-beasley para seleccionar a la mejor hormiga, en el proceso se envía una

por una y el algoritmo evalúa los resultados y selecciona a la mejor, realizar el proceso de mutación y luego comparar con el resto determinando si hubo mejora o no. Esta comparación se hace mediante el fitness o función objetivo. Para concluir se hacen las iteraciones necesarias hasta encontrar un punto de convergencia.

Crossover de Algoritmo Memético y Búsqueda local Guiada

Diana Holstein (1998) realizó un proceso en donde combinó un Algoritmo Memético (AM) con uno heurístico llamado Búsqueda Local Guiada (GLS) para resolver el TSP y verificar si había algún cambio respecto a la solución del mismo en un AG. Un algoritmo Memético es una variante de un algoritmo genético el cual se basa en la población, este algoritmo incorpora la mayor cantidad de conocimiento del dominio posible durante el proceso de generación de una nueva población. Por otro lado, la GLS aprovecha información relacionada con la estructura del problema y el curso de la búsqueda para guiar la exploración de un espacio [16].

Para el algoritmo creado, se parte de un AM tomando soluciones validas de un TSP, se intenta hacer mejoras eligiendo a las soluciones mas aptas y generando nuevas con cada iteración. En cada iteración se evalúan las soluciones, se seleccionan las que sirven para hacer otra población y se hacen recombinaciones para adicionar diversidad al algoritmo. El GLS en este caso se usa con el objetivo de hallar óptimos locales, esta heurística genera una técnica para incluir métodos de "penalización", esto significa que el algoritmo cambia los parámetros de forma automática cuando ve que se ha quedado atrapado en un óptimo local para "destrabar" el algoritmo.

Algoritmo de colonia de hormigas y temperatura

Alejandro Torres, Benjamín Arenas y Rodrigo vidal (2000) realizaron un ACO con una modificación basada en la temperatura para solucionar un TSP en un grafo de tipo Euclidiano. La idea era tomar elementos de un algoritmo *Simulated annealing* dentro de la cadena de decisión de movimiento de las hormigas, en dicho caso se establece un criterio para dictar de manera aleatoria la exploración de la hormiga.

Si la probabilidad asociada a una ciudad j seleccionada como el próximo movimiento es menos que un número aleatorio, entonces la nueva selección se asignará de forma aleatoria desde un vecindario factible de los nodos adyacentes. A medida que el algoritmo genera probabilidades mayores de movimien-

tos por un tour encontrado la probabilidad de asignación de un nuevo movimiento aleatorio es menor; esta es la analogía que se desea recrear con el concepto de temperatura.

Algoritmos de memoria

José Ramos (2015) realizó su trabajo de maestría centrado en solucionar el TSP con ayuda de un AG y El aplicativo de Google maps, en este caso, él habla de dos tipos de algoritmos metaheurísticos trabajados en la historia, los de poca memoria y los basados en memoria. Entre los algoritmos basados en poca memoria, destaca el *Simulated annealing* ya mencionado anteriormente, el cual está basado en la temperatura. El autor expresa que Tarantilis, uno de sus principales expositores presentó variantes de dicho algoritmo en los años 2001, 2002 y 2004. Por otro lado, el autor nombra los algoritmos de memoria; este nombre fue acuñado para denominar a los algoritmos de búsqueda Tabú y los algoritmos basados en memoria adaptativa. Igualmente, en dicho conjunto están los algoritmos genéticos desarrollados por Holland, el de colonia de hormigas por Dorigo, los de enjambre de partículas por Kennedy, Eberhart y los algoritmos Meméticos; todos los anteriormente nombrados se destacan por tener estructuras que se pueden considerar como recuerdos.

4.4. Programación Matemática

La Programación Matemática es un campo amplio y moderno de estudio que se enfoca en el diseño de metodologías prácticas que sirvan para resolver problemas de optimización. Con frecuencia, estos problemas surgen en diversos contextos donde la toma de decisiones juega un papel fundamental. Con la programación matemática se busca encontrar un conjunto de valores de variables que optimicen una función objetivo, sujeta a un conjunto de restricciones utilizando enfoques teóricos y computacionales.

Siempre se tendrá la intención de maximizar o minimizar una función, dependiendo de esto, se pueden incluir una amplia gama de problemas, cuyo objetivo es buscar optimalidad en relación a la determinación de alguna solución. Así, la optimización puede definirse como: las técnicas a seguir para maximizar o minimizar la respuesta a un sistema. Por lo general, dicha respuesta suele tener indicadores de tipo costo, producción, ganancia, etc.

De igual forma, la respuesta se denomina objetivo y la función asociada al problema como función objetivo [5]. Asimismo las "técnicas.º "políticas", son un conjunto de valores que toman los factores que

se pueden controlar a fin de regular el rendimiento del sistema, son básicamente variables independientes de la función las cuales se pueden alterar para buscar una respuesta esperada. Para diseñar un modelo de optimización que dé solución a cualquier tipo de problemas, se deben contar con los siguientes tres elementos:

- **Las variables de decisión:** son las que determinan las posibles decisiones que pueden tomarse en el sistema entorno a la solución, por lo general estas variables son independientes y de carácter cuantitativo.

- **Restricciones:** son las que especifican el conjunto de valores de las variables de decisión que se admiten dentro del sistema, en ocasiones las restricciones dependen de la naturaleza de los elementos y sus relaciones, de las aplicaciones reales del problema, de las limitaciones de la misma función, entre otras.

- **Función objetivo:** es la que establece el coste/beneficio asociado a cada decisión que, relacionado con la técnica de optimización adecuada y la o las restricciones del problema, permitirá asignar a cada grupo posible de valores de las variables de decisión su valor de coste/beneficio para seleccionar el óptimo.

En [26] se describe la estructura general un problema de optimización matemáticamente de la siguiente manera:

$$\text{Minimizar } x \in R^n \qquad f_i(x), \qquad (i = 1, 2, 3, ..., M)$$

$$\text{Sujeto a } \phi_j(x) = 0, (j = 1, 2, 3, ..., J)$$

$$\psi_k(x) \leq 0 \qquad (k = 1, 2, 3, ..., K)$$

Donde $f_i(x)$, $\phi_j(x)$ y $\psi_k(x)$ son funciones del vector designado

$$x = (x_1, x_2, ..., x_n)^T$$

De estas expresiones, los componentes x_i de x son llamados variables de decisión, los cuales toman valores del conjunto de los números reales. Las funciones $f_i(x)$ donde $i = 1, 2, 3, ..., M$, son llamadas funciones objetivo, si $M = 1$ se dice que solo hay un objetivo. El objetivo de la función usualmente se asocia a algún costo o energía.

Clasificación de problemas de optimización

En [5] se da una clasificación de los diferentes tipos de problemas de optimización, los cuales son:

- **Optimización continua:** un problema se dice que es de optimización continua cuando todas las variables de decisión pueden tomar valores en el conjunto de los reales.
- **Programación lineal:** esta clase de problemas surgen cuando la función objetivo y las restricciones son lineales.
- **Optimización combinatoria:** este tipo de problemas solo se aplica cuando se trabaja con variables discretas (solo toman valores enteros).
- **Optimización mixta:** problemas de optimización donde se pueden presentar en un mismo ejercicio instancias con variables continuas y discretas.

5. Desarrollo del trabajo

En el siguiente capítulo se presenta una visión general de cómo abordar el TSP de manera convencional, sin la implementación de programas computacionales. Para lograrlo, se examinarán tres grafos diferentes (de 4, 5 y 6 nodos), se construirá su matriz de adyacencia y se revisarán todos los recorridos posibles hasta establecer el de menor coste o distancia, en cada caso. Los grafos mostrados para cada ejemplo, por comodidad, serán todos completos y no dirigidos.

En el siguiente apartado, se explica el paso a paso de un ACO adaptado del clásico para entender conceptualmente su desarrollo y funcionamiento con un grafo de 4 nodos. De igual forma, se comparan los resultados obtenidos por este algoritmo junto con los algoritmos de Fuerza Bruta y Held-Karp.

5.1. Ejemplos para el TSP:

Un proceso que se puede seguir para abordar el TSP consiste en:

1. Representar el problema como un grafo completo. Las ciudades serán los nodos y las distancias entre ellos serán las aristas. Para cada par de nodos habrá una distancia asociada.

2. Construir la matriz de adyacencia para representar las distancias.

3. Seleccionar un nodo cualquiera como inicio y buscar todas las rutas posibles. Únicamente funcionan las rutas que visitan cada nodo exactamente una vez y regresan al nodo inicial, esto es, aquellas rutas que son ciclos hamiltonianos.

4. Calcular la distancia total de cada ciclo hamiltoniano sumando los valores correspondientes a cada arista del grafo.

5. Comparar todos los resultados del paso anterior y escoger el de menor valor. La ruta escogida será la óptima.

5.1.1. Ejemplo con 4 ciudades

Siguiendo los pasos anteriormente descritos:

1. Representar:

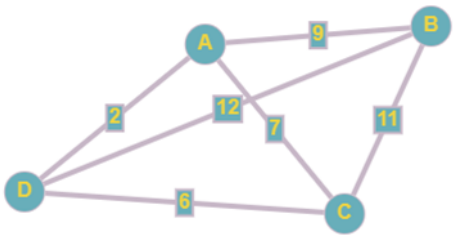


Figura 5.1: Ejemplo grafo 4 nodos

2. Construir

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 9 & 7 & 2 \\ 9 & 0 & 11 & 12 \\ 7 & 11 & 0 & 6 \\ 2 & 12 & 6 & 0 \end{pmatrix} \end{array}$$

3. Seleccionar:

Iniciando desde el nodo B, las siguientes rutas son ciclos hamiltonianos:

Figura 5.2: Caminos 1 y 2

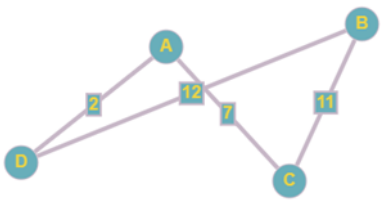


Figura 5.3: Camino 3

4. Calcular:

Ruta 1: B-C-D-A-B=11+6+2+9=28

Ruta 2: B-A-C-D-B=9+7+6+12=34

Ruta 3: B-D-A-C-B=12+2+7+11=32

5. Comparar:

Ruta 1: B-C-D-A-B=11+6+2+9=28

Entre todas las rutas la de menor valor, la óptima, es la ruta 1 con una distancia total de 28. En el caso de 4 ciudades la cantidad de ciclos hamiltonianos que se obtienen solamente son tres. Aquí, la verificación y comparación manual ruta a ruta es relativamente sencilla.

5.1.2. Ejemplo 2 con 5 ciudades

1. Representar:

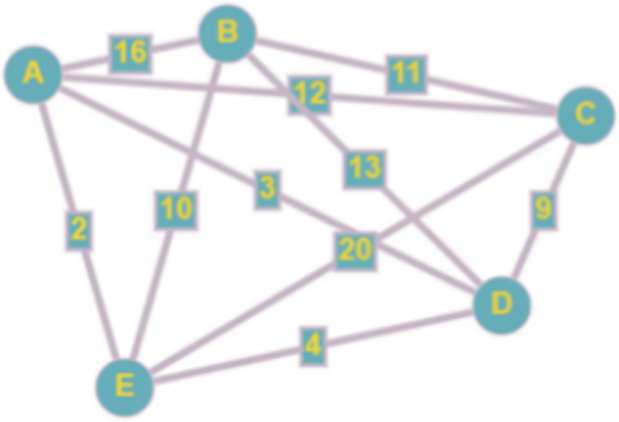


Figura 5.4: Ejemplo grafo 5 nodos

2. Construir:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 16 & 12 & 3 & 2 \\ 16 & 0 & 11 & 13 & 10 \\ 12 & 11 & 0 & 9 & 20 \\ 3 & 13 & 9 & 0 & 4 \\ 2 & 10 & 20 & 4 & 0 \end{pmatrix} \end{array}$$

3. Seleccionar:

Iniciando desde el nodo A, las siguientes rutas son ciclos hamiltonianos:

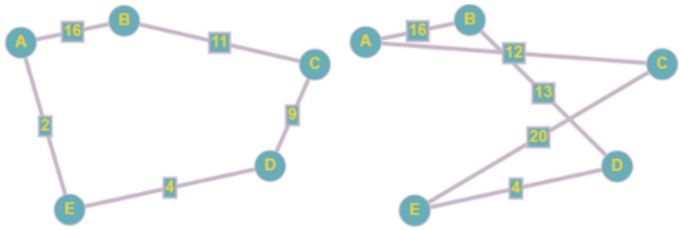


Figura 5.5: Camino 1 y 2

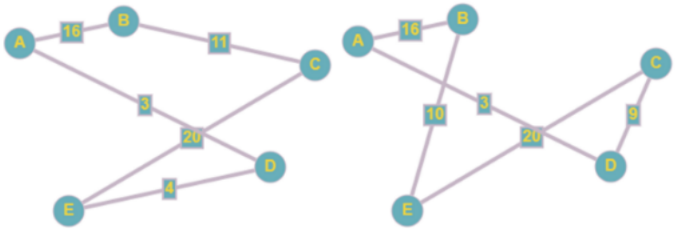


Figura 5.6: Camino 3 y 4

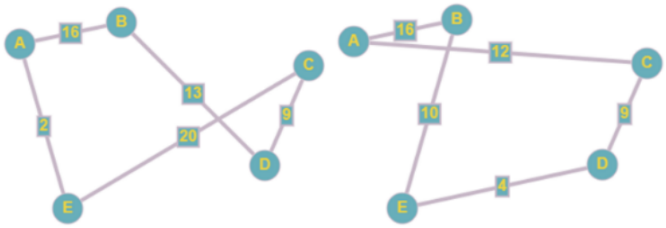


Figura 5.7: Camino 5 y 6

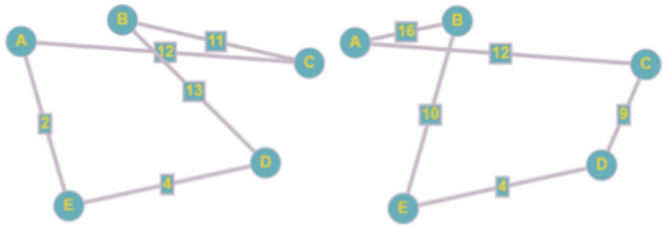


Figura 5.8: Camino 7 y 8

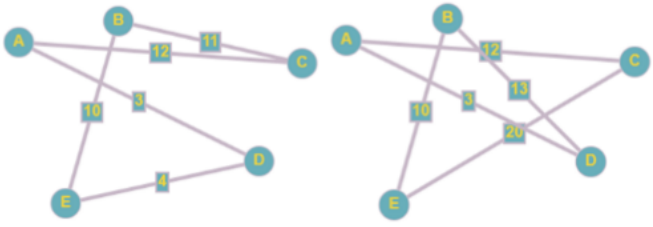


Figura 5.9: Camino 9 y 10

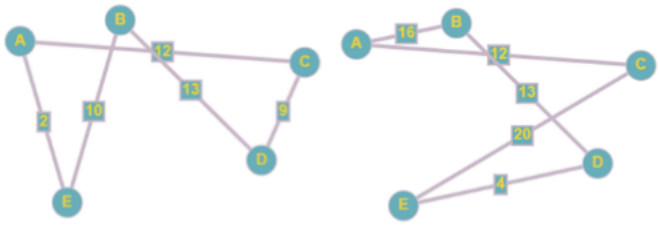


Figura 5.10: Camino 11 y 12

4. Calcular:

Ruta 1: $A-B-C-D-E-A=16+11+9+4+2=42$

Ruta 2: $A-B-C-E-D-A=16+11+3+20+4=54$

Ruta 3: $A-B-D-C-E-A=16+13+9+20+2=60$

Ruta 4: $A-B-D-E-C-A=16+12+13+20+4=65$

Ruta 5: $A-B-E-C-D-A=16+10+20+9+3=58$

Ruta 6: $A-B-E-D-C-A=16+10+4+9+12=51$

Ruta 7: $A-C-B-D-E-A=12+11+13+4+2=42$

Ruta 8: $A-C-B-E-D-A=12+11+10+4+3=40$

Ruta 9: $A-C-D-B-E-A=12+9+13+10+2=46$

Ruta 10: $A-C-D-E-B-A=12+9+4+10+16=51$

Ruta 11: $A-C-E-B-D-A=12+20+10+13+3=58$

Ruta 12: $A-C-E-D-B-A=12+20+4+13+16=65$

5. Comparar:

Ruta 8: $A-C-B-E-D-A=12+11+10+4+3=40$

Aquí, la ruta 8 es la óptima con una distancia total de 40. Nótese que en el caso de 5 ciudades la cantidad de rutas posibles que son ciclos hamiltonianos aumenta considerablemente. ¿Que pasaría si se añade otra ciudad? A priori, podría pensarse que es posible resolverlo siguiendo los mismos pasos, y aunque lo es, el siguiente ejemplo mostrará que sucede.

5.1.3. Ejemplo 3: 6 ciudades

Al seguir el mismo procedimiento de los ejemplos anteriores, los primeros dos pasos (representar y construir) se pueden realizar sin ningún inconveniente, contrario a lo que sucede con los pasos posteriores.

Representar:

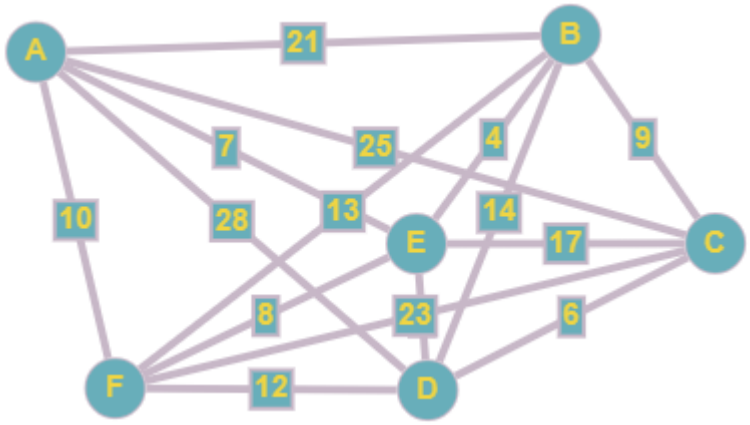


Figura 5.11: Ejemplo grafo 6 nodos

Construir

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 21 & 25 & 28 & 7 & 10 \\ 21 & 0 & 9 & 14 & 4 & 13 \\ 25 & 9 & 0 & 6 & 17 & 23 \\ 28 & 14 & 6 & 0 & 3 & 12 \\ 7 & 4 & 17 & 3 & 0 & 8 \\ 10 & 13 & 23 & 12 & 8 & 0 \end{pmatrix} \end{array}$$

Seleccionar:

Mientras que en los grafos completos de 4 y 5 nodos, la cantidad de rutas que cumplían con la condición de ser ciclos hamiltonianos era reducida, a medida que se añaden más ciudades al problema, el número de ciclos aumenta significativamente. En la tabla 1.6 del capítulo anterior, se puede apreciar que con tan solo 6 ciudades, existen un total de 60 ciclos hamiltonianos posibles.

Este incremento exponencial hace que encontrar la ruta óptima, ya sea de forma manual o computacional, se convierta en una tarea más desafiante y compleja que requerirá de un mayor esfuerzo

y dedicación. Por consiguiente, se vuelve necesario implementar otros procedimientos que aborden el problema de manera más sencilla y que permitan obtener soluciones viables en un menor tiempo.

Una alternativa que se puede considerar para abordar el TSP cuando la cantidad de ciudades es igual o mayor a 6 consiste en implementar el Algoritmo de Optimización por Colonia de Hormigas (ACO). Este algoritmo proporciona un enfoque más eficiente y simplificado que facilita la resolución del problema, ya sea de forma manual o computacional.

5.2. Estructura del ejemplo con ACO

Como se señaló en el capítulo anterior, el algoritmo de optimización por colonia de hormigas (ACO) es una estrategia meta-heurística inspirada en el comportamiento de hormigas reales que buscan los mejores caminos entre fuentes de alimento y su colonia. La idea fundamental de este ejemplo es abstraer el comportamiento de búsqueda y encontrar, en lo posible, el camino más corto para un grafo completo con 6 nodos.

En particular, la búsqueda y establecimiento de la mejor solución solo puede evidenciarse una vez se itere el algoritmo en repetidas ocasiones. Sin embargo, por comodidad en el desarrollo de este ejemplo, las iteraciones que se realicen y las hormigas que se utilicen no serán tantas dado que el objetivo es recrear el proceso de búsqueda de soluciones y mostrar el funcionamiento manual del algoritmo.

5.2.1. Procedimiento:

Inicialmente, se representa el problema como un grafo completo no dirigido indicando sus nodos, sus aristas y la distancia asociada a cada una de estas como en la 5.11. Luego se establece la cantidad de iteraciones y de hormigas. Como en este caso el grafo tiene 15 caminos en total (cada uno con su respectiva distancia) y 60 ciclos hamiltonianos posibles, solamente se utilizarán 2 hormigas y se iterará el algoritmo una vez, contrario a su implementación computacional.

Posteriormente, se calcula el componente de información heurística (o visibilidad) junto con un número de feromonas inicial en cada uno de los caminos: por practicidad se utiliza un valor pequeño, en este caso 0,1. La tabla 5.1 recopila la información mencionada hasta el momento:

Camino	Distancia	Visibilidad (η)	Feromona (τ)
A-B	21	1/21	0,1
A-C	25	1/25	0.1
A-D	28	1/28	0,1
A-E	7	1/7	0,1
A-F	10	1/10	0,1
B-C	9	1/9	0,1
B-D	14	1/14	0,1
B-E	4	1/4	0,1
B-F	13	1/13	0,1
C-D	6	1/6	0,1
C-E	17	1/17	0,1
C-F	23	1/23	0,1
D-E	3	1/3	0,1
D-F	12	1/12	0,1
E-F	8	1/8	0,1

Cuadro 5.1: Información

Para iniciar con la búsqueda de algún camino que sea ciclo hamiltoniano, se ubican ambas hormigas en cualquiera de los nodos y se verifica a cuáles otros podrán dirigirse a continuación. Para este ejemplo, si las hormigas inician en el nodo A podrán ir hacia los nodos B, C, D, E o F, respectivamente. Para saber hacia cuál de todos irán es necesario calcular la probabilidad de elección utilizando la ecuación expuesta en el capitulo anterior.

Realizando los cálculos correspondientes para ambas hormigas, se observa que tienen las mismas probabilidades de elegir cualquiera de los nodos cercanos, salvo que esta elección dependerá completamente del azar. Como la suma de las probabilidades siempre será igual a 1, una estrategia que se puede seguir para determinar el camino que cada hormiga escogerá, consiste en ponderar todas las probabilidades en un intervalo entre 0 y 1, seleccionar un número aleatorio cualquiera y, dependiendo de este, establecer a cuál de las probabilidades anteriormente calculadas está más próxima.

Para generar números aleatorios entre 0 y 1, se utilizó una calculadora *Cassio* y su función *Ran#*. El primer número generado para cada hormiga fue: 0,089 y 0,178. Es decir, por probabilidad, la hormiga 1 decidió ir hacia el nodo B y la hormiga 2 hacia el nodo C.

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
A-B	0,0047	0,1291
A-C	0,004	0,1098
A-D	0,0035	0,0961
A-E	0,0142	0,3901
A-F	0,01	0,2747
$\sum$	0,0364	≈ 1

Cuadro 5.2: Primera elección

Ubicadas ambas hormigas en sus respectivos nodos, como cada una escogerá el siguiente de manera aleatoria, aunque el procedimiento es exactamente el mismo, de aquí en adelante los resultados se mostraran en dos tablas diferentes.

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
B-C	0,0111	0,2185
B-D	0,0071	0,1397
B-E	0,025	0,4921
B-F	0,0076	0,1496
$\sum$	0,0508	≈ 1

Cuadro 5.3: Camino desde B, hormiga 1

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
C-B	0,0111	0,2936
C-D	0,0166	0,4391
C-E	0,0058	0,1534
C-F	0,0043	0,1137
$\sum$	0,0378	≈ 1

Cuadro 5.4: Camino desde C, hormiga 2

Generando los números aleatorios: 0,912 y 0,307, quiere decir que, por probabilidad, la hormiga 1 decidió ir hacia el nodo F mientras que la hormiga 2 hacia el nodo D.Ahora, ubicadas cada una en sus nodos:

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
F-C	0,0043	0,1713
F-D	0,0083	0,3306
F-E	0,0125	0,4980
$\sum$	0,0251	≈ 1

Cuadro 5.5: Camino desde F, hormiga 1

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
D-B	0,0071	0,1457
D-E	0,0333	0,6837
D-F	0,0083	0,1704
$\sum$	0,0487	≈ 1

Cuadro 5.6: Camino desde D, hormiga 2

Después de generar los números aleatorios: 0,188 y 0,853, la hormiga 1 irá hacia el nodo C y la hormiga 2 hacia el nodo F. Desde estos, a cada una le quedan solamente dos opciones: la hormiga 1 podrá dirigirse a D o E y la hormiga 2 a B o E.

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
C-D	0,0166	0,7410
C-E	0,0058	0,2589
$\sum$	0,0224	≈ 1

Cuadro 5.7: Camino desde C, hormiga 1

Caminos	$\tau(e)\eta(e)$	$P_{x,y}$
F-B	0,0076	0,3781
F-E	0,0125	0,6218
$\sum$	0,0201	≈ 1

Cuadro 5.8: Camino desde F, hormiga 2

Generando los números aleatorios: 0,968 y 0,275 se determina, por probabilidad, que la hormiga 1 irá al nodo E mientras que la hormiga 2 al nodo B. Finalmente, a ambas hormigas solo les quedan dos alternativas: volver a donde iniciaron o dirigirse al nodo restante y completar el ciclo. Por conveniencia en el desarrollo del ejemplo, cada hormiga se dirigirá hacia el nodo restante y luego regresará al inicio. De cualquier forma, en un algoritmo codificado la elección de la hormiga es completamente aleatoria por lo que no necesariamente tendrá que dirigirse al nodo restante sino que podrá volver al inicio y en las iteraciones siguientes ir estableciendo el camino hasta que pase por todos los nodos.

En conclusión, tras aplicar el mismo procedimiento con dos hormigas el recorrido de cada una, iniciando y terminando en el mismo nodo fue:

- **Hormiga 1:**

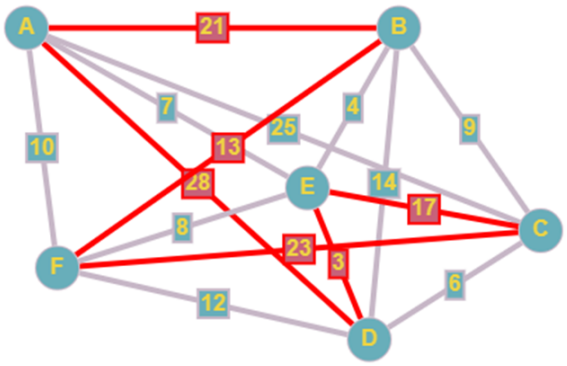


Figura 5.12: Camino recorrido por la Hormiga 1

Camino: $A - B - F - C - E - D$

Distancia total: $21 + 13 + 23 + 17 + 3 + 28 = 105$

- **Hormiga 2:**

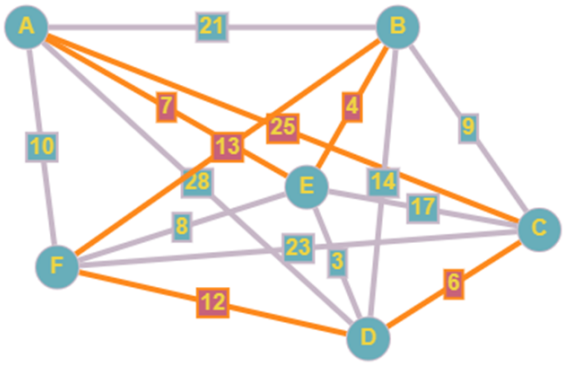


Figura 5.13: Camino recorrido por la Hormiga 2

Camino: $A - C - D - F - B - E$

Distancia total: $25 + 6 + 12 + 13 + 4 + 7 = 67$

Como se puede observar, entre los dos caminos el más corto hasta el momento es el camino recorrido por la hormiga 2 con una distancia total de 67, pero este no puede seleccionarse todavía ya que es necesario actualizar las feromonas para ver en cuál camino se comienza a acumular. Entre más se itere el algoritmo más feromonas se irán acumulando lo que progresivamente resultará en el camino óptimo.

La actualización de las feromonas se realiza siguiendo la ecuación mostrada en el capitulo anterior. Para este caso, la taza de evaporación ρ será igual a 0,01 y la constante de actualización de la feromona (aprendizaje) Q a 1; en ambos casos puede ser cualquier valor. Los cálculos realizados se organizarán en la siguiente tabla en donde la tercer y cuarta columna corresponden a la cantidad de feromonas depositadas por la primera y segunda hormiga en cada camino transitado:

Camino	$(1-\rho)$	$\Delta\tau_x y^1$	$\Delta\tau_x y^2$	Feromona (τ)
A-B	0,099	0,0104	0	0,1094
A-C	0,099	0	0,0149	0,1139
A-D	0,099	0,0104	0	0,1094
A-E	0,099	0	0,0149	0,1139
A-F	0,099	0	0	0,099
B-C	0,099	0	0	0,099
B-D	0,099	0	0	0,099
B-E	0,099	0	0,0149	0,1139
B-F	0,099	0,0104	0,0149	0,1243
C-D	0,099	0	0,0149	0,1139
C-E	0,099	0,0104	0	0,1094
C-F	0,099	0,0104	0	0,1094
D-E	0,099	0,0104	0	0,1094
D-F	0,099	0	0,0149	0,1139
E-F	0,099	0	0	0,099

Cuadro 5.9: Actualización de feromonas

En definitiva, la actualización de feromonas (τ) para cada uno de los caminos aumenta o disminuye según las hormigas los transiten o no. Después de la actualización, el algoritmo vuelve a iterarse con el nuevo valor para la feromona y a medida que se repita el proceso más y más veces, en algún momento la feromona de los caminos menos transitados será 0 lo que hará que algunos "desaparezcan"para las hormigas y converja en uno solo.

5.3. Construcción de algoritmos

A continuación, se explica el funcionamiento del ACO desarrollado y adaptado para un grafo de cuatro nodos en el cual se darán valores arbitrarios con tal de entender conceptualmente lo que se espera realizar en el algoritmo. Además, se muestran y comparan los resultados con otros dos algoritmos propuestos: el algoritmo de fuerza bruta y el algoritmo Held-Karp, en diferentes grafos.

5.3.1. ACO en nodos distintos

En el caso del ACO propuesto, cada hormiga recorrerá el grafo desde una ciudad distinta a comparación al ACO clásico donde todas las hormigas parten desde un mismo nodo inicial. Es así que para dicho ejemplo se tendrá en cuenta la siguiente estructura para conseguir el camino hamiltoniano más corto:

1. **Inicio:**

   Se selecciona un conjunto de hormigas para iniciar desde nodos aleatorios en el grafo. Cada hormiga se coloca en un nodo diferente como punto de partida.

2. **Construcción de soluciones parciales:**

   Cada hormiga construye una solución parcial eligiendo su próximo nodo de manera probabilística. La elección se realiza considerando la feromona depositada en las aristas, una heurística (visión de la hormiga) que mide la conveniencia de visitar un nodo en particular, También un escalar llamado "importancia de la heurística" el cual entre más grande sea indicará que tanto peso tendrá la heurística para escoger la próxima ciudad y en este ejemplo será 3, de la misma manera la feromona tendrá un "factor de importancia" que indicará el peso de la feromona al hallar la probabilidad, para este ejemplo tomará un valor de 1.

3. **Movimiento de las hormigas:**

   Las hormigas se mueven de nodo en nodo hasta que cada una haya completado una solución completa. Esto significa que cada hormiga haya visitado cada nodo exactamente una vez sin repeticiones.

4. **Evaluación de soluciones:**

   Se evalúa cada solución construida por las hormigas. Esto es, calcular la longitud (coste) total del camino. Cuanto mejor sea la solución, más feromona se depositará en las aristas visitadas por la hormiga.

5. **Actualización de feromonas:**

   Se actualizan las cantidades de feromona en las aristas de acuerdo con las soluciones encontradas. Las aristas que forman parte de soluciones de las hormigas recibirán una mayor cantidad de feromona, mientras que las aristas menos útiles tendrán menos feromona.

6. **Evaporación de feromonas:**

   Las cantidades de feromona en todas las aristas se reducen gradualmente mediante un proceso de evaporación, esto simula la pérdida natural de feromona con el tiempo de las aristas. La tasa de evaporación será de 0.3 para nuestro ejemplo.

7. **Criterio de parada:**

   Se verifica si se cumple un criterio de parada, en nuestro caso será un número de iteraciones.

8. **Actualización de soluciones globales:**

   Si se encuentra una solución mejor durante una iteración, se actualiza la mejor solución global conocida.

9. **Retorno al paso 2:**

   Si todavía no se han completado la cantidad de iteraciones, las hormigas regresan al paso 2 y repiten el proceso. este ciclo se repite hasta que se cumple el número de iteraciones. La idea es que con cada iteración las soluciones converjan a un número el cuál representará la distancia de la solución más óptima del algoritmo.

Su diagrama de flujo se representa a continuación:

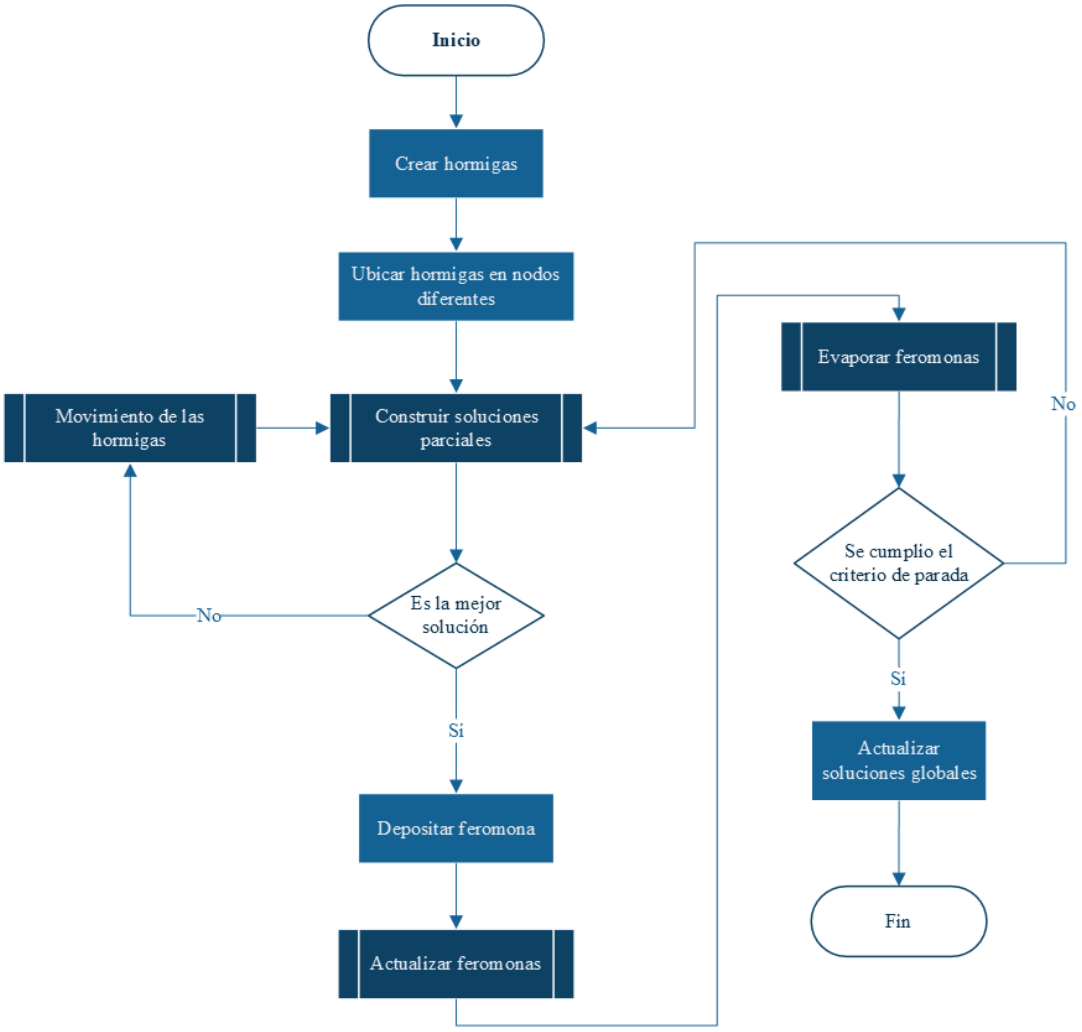


Figura 5.14: Diagrama de flujo ACO.

5.3.2. Ejemplo:

Supongamos que tenemos el siguiente grafo 5.15 y queremos encontrar el camino hamiltoniano mas corto que existe en dicho grafo.

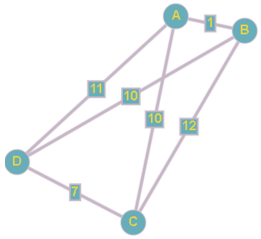


Figura 5.15: Grafo de cuatro nodos

En primera instancia se establece la matriz de distancias que indica el camino más corto que

hay entre cada par de nodos:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 10 & 11 \\ 1 & 0 & 12 & 10 \\ 10 & 12 & 0 & 7 \\ 11 & 10 & 7 & 0 \end{pmatrix} \end{array}$$

También es necesario crear una matriz de feromona para cada arista del grafo. En este ACO se tendrá en cuenta una feromona inicial entre cada arista que conecta un par de nodos de 1, de tal manera que, por cada iteración, dicha feromona aumente si alguna hormiga pasa por dicho camino o disminuya en el caso contrario. Así mismo se reducirá la feromona por la tasa de evaporación arbitraria que se coloque en el algoritmo.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Paso 1:

Creamos 4 listas las cuales representan los caminos que tomarán cada una de las hormigas en la primera iteración, como cada hormiga empieza desde una ciudad distinta. Es necesario colocar como primer elemento de cada lista una ciudad de las cuatro existentes de manera aleatoria de la siguiente manera: (A), (B), (C) y (D)

Paso 2:

Tomaremos el primer camino, y en otra lista a parte, escribiremos todas las ciudades por la cual la hormiga no ha viajado. En nuestro caso serán las siguientes: (B C D)

La intención de crear esta nueva lista es analizar el posible nodo al cual la hormiga puede visitar en seguida del anterior y como se expresó anteriormente se tomarán los siguientes criterios para que la hormiga escoja el siguiente camino:

Heurística (visión): La heurística es un número que representa que tan lejos está otra ciudad desde el nodo donde la hormiga está ubicada, este número influye en la probabilidad de escoger la siguiente ciudad del grafo.

$$H = \frac{1}{matrizdedistancias}$$

Donde H es la Heurística y matriz de distancias la matriz que teníamos anteriormente, H también es una matriz, en la cual, cada uno de sus elementos se hallarán al tomar el número 1 y dividirlo entre la distancia existente entre cada par de nodos.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 10 & 11 \\ 1 & 0 & 12 & 10 \\ 10 & 12 & 0 & 7 \\ 11 & 10 & 7 & 0 \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 0,1 & 0,09 \\ 1 & 0 & 0,083 & 0,1 \\ 0,1 & 0,083 & 0 & 0,14 \\ 0,09 & 0,1 & 0,14 & 0 \end{pmatrix} \end{array}$$

Se puede observar que entre mas cercano sea el nodo, el valor de la heurística es mucho más alto, esto influirá en gran medida al momento en que la hormiga tome su siguiente camino.

En seguida crearemos una relación proporcional entre la heurística y la feromona que existe en cada camino creando una suma que llamaremos FA o feromona acumulada. Para hallar FA haremos en siguiente proceso:

Con el camino: (A) y las ciudades sin visitar (B C D) Tomamos la feromona que hay en el camino de A-B la cuál es 1 (cómo se ve en la matriz de feromonas y porque es la iteración inicial) y la elevamos al factor de importancia de la feromona (1), este valor lo multiplicamos por la heurística elevado a su factor de importancia que es 3:

$$1^1 \left(\frac{1}{1}\right)^3 = 1$$

Hacemos lo mismo con A-C y A-D, cuyas soluciones respectivas son:

$$1^1 \left(\frac{1}{10}\right)^3 = 0,001$$

$$1^1 \left(\frac{9}{100}\right)^3 = 0{,}000729$$

Los números 0.1 y 0.9 de la matriz de heurística se escriben en fracción porque así es más evidente observar el peso del factor de importancia de la heurística. Se observa entonces que al reemplazar en las fracciones el denominador es más grande que el numerador, por tal motivo al elevar la heurística a la potencia 3, dará como resultado un número mucho más pequeño. Esto puede interpretarse como: si el nodo está más lejos el resultado será mucho más pequeño y por lo tanto la probabilidad que una hormiga escoja dicho camino será pequeña también.

$$FA = 1 + 0{,}001 + 0{,}000729$$

$$FA = 1{,}001729$$

Con este valor (FA) podemos encontrar la probabilidad para que la hormiga escoja el siguiente camino de la siguiente manera:

Para el camino de A-B tomaremos nuevamente la feromona que hay actualmente en el camino, la elevamos a la 1 (factor importancia de feromona) y la multiplicaremos por la heurística elevada a su factor de importancia dividida entre FA

$$P_{AB} = 1^1 \cdot \frac{1^3}{1{,}001729} = 0{,}999984364$$

Este valor lo anotamos en una lista a parte llamada P de probabilidad

$$P = [0{,}999984364]$$

Hacemos lo mismo con el resto de los caminos A-C y A-D

$$P_{AB} = 1^1 \cdot \frac{0,1^3}{1{,}001729} = 0{,}000982739$$

$$P_{AB} = 1^1 \cdot \frac{0,1^3}{1{,}001729} = 0{,}00007277417$$

Después de actualizar la lista de probabilidades nos quedaría de la siguiente manera:

$$P = [0{,}999984364 \quad 0{,}000982739 \quad 0{,}00007277417]$$

Donde el primer elemento es la probabilidad que la hormiga tome el camino desde A hasta B, el segundo elemento que vaya desde A hasta C y el tercero que vaya desde A hasta D además que la suma de cada elemento de la lista tiene que dar como resultado 1:

$$0{,}999984364 + 0{,}000982739 + 0{,}00007277417 = 1$$

Es evidente a estas alturas que la hormiga tomará el camino de A hacia B dado que hay un 99.9 % de probabilidad que tome dicho camino contra 0.098 % y 0.007 % de los otros dos y si se piensa, tiene sentido dado que de manera inicial la cantidad de feromonas que hay en cada camino es igual y el nodo B está a 1 unidad de distancia desde A a comparación de los caminos C y D que se encuentran a 10 y 11 unidades respectivamente.

Continuando con el proceso al camino inicial se le agrega el nodo B y nuestro nuevo camino sería: (A B). Entonces debemos repetir el procedimiento nuevamente pero ahora partiendo desde B y las ciudades sin visitar serán C-D dado que A era el lugar de partida de la hormiga. Al realizar el mismo procedimiento se determina el orden de las 4 ciudades que tomará la hormiga siendo este: (A B D C).

Se repite el proceso con cada uno de los caminos hasta obtener el camino de cada hormiga

(D C B A)

(C D B A)

(B C D A)

En el paso siguiente se calcula la distancia de cada camino sumando la distancia nodo a nodo con base a la matriz de distancias, al llegar al último nodo se debe sumar la distancia desde dicho nodo

hasta el primero para completar un ciclo hamiltoniano.

- Distancia primer camino:

  $Dist1 = Dist(A,B) + Dist(B,D) + Dist(D,C) + Dist(C,A) = 1 + 10 + 7 + 10 = 28$

- Distancia segundo camino:

  $Dist2 = Dist(D,C) + Dist(C,B) + Dist(B,A) + Dist(A,D) = 7 + 12 + 1 + 11 = 31$

- Distancia tercer camino:

  $Dist3 = Dist(C,D) + Dist(D,B) + Dist(B,A) + Dist(A,C) = 7 + 10 + 1 + 10 = 28$

- Distancia cuarto camino:

  $Dist4 = Dist(B,C) + Dist(C,D) + Dist(D,A) + Dist(A,B) = 12 + 7 + 11 + 1 = 31$

Después de encontrar la distancia de cada viaje se selecciona el que tenga la distancia mas corta y esta será la solución mas óptima en la primera instancia, si dos caminos son iguales en magnitud como la más pequeña se selecciona uno al azar y se sigue con el procedimiento. Como ya sabemos el camino que realiza cada hormiga, podemos actualizar la feromona de cada arista entre nodo, para ello nos dirigimos a la matriz de feromona:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Tomamos el primer camino (A B D C) y hacemos el siguiente análisis: del nodo A la hormiga se dirige a B por lo tanto viajará por dicha arista, en el momento que la hormiga realiza el recorrido, dicha arista tiene una cantidad de feromona de 1, a esta cantidad se le suma la feromona que deposita la hormiga cuya cantidad será:

$$CDF = \frac{1}{Dtr}$$

Donde CFD es la cantidad de feromona depositada y Dtr es la distancia recorrida en el camino, este valor se agrega a la feromona actual y de esta manera se actualiza la feromona de las hormigas que transitan el camino.

$$CDF = \frac{1}{28} \approx 0,035714$$

Feromona depositada en cada camino

- Camino 1: $A, B = 1 + 0,035714 = 1,035714$
- Camino 2: $B, D = 1 + 0,035714 = 1,035714$
- Camino 3: $D, C = 1 + 0,035714 = 1,035714$
- Camino 4: $C, A = 1 + 0,035714 = 1,035714$

$$\begin{pmatrix} 1 & 1,035714 & 1 & 1 \\ 1 & 1 & 1 & 1,035714 \\ 1 & 1 & 1 & 1 \\ 1,035714 & 1 & 1 & 1 \\ 1 & 1 & 1,035714 & 1 \end{pmatrix}$$

Se hace el resto del procedimiento de la misma manera, ahora a la feromona actual se le agreda el CFD, si hacemos el continuamos con el resto de los caminos la matriz resultante debe ser la siguiente:

$$\begin{pmatrix} 1 & 1,71428 & 1 & 1,035714 \\ 1,71428 & 1 & 1 & 1,71428 \\ 1 & 1 & 1 & 1 \\ 1,035714 & 1,035714 & 1 & 1,71428 \\ 1,035714 & 1,035714 & 1,71428 & 1 \end{pmatrix}$$

Por último, se aplica la evaporación de la feromona, esta será una evaporación que se aplica a cada camino después que cada hormiga realiza su recorrido, para este ejercicio será de 0.1 y dicho valor se reduce a cada elemento de la matriz de feromona.

$$\begin{pmatrix} 0,9 & 1,61428 & 0,935714 & 0,935714 \\ 1,61428 & 0,9 & 0,9 & 1,61428 \\ 0,9 & 0,9 & 0,9 & 0,9 \\ 0,935714 & 0,935714 & 0,9 & 1,61428 \\ 0,935714 & 0,935714 & 1,61428 & 0,9 \end{pmatrix}$$

Cuando actualizamos la feromona, decimos que terminamos una iteración, en este punto tenemos un posible camino mas eficiente

$$(A \quad B \quad D \quad C) = 28$$

Dicho camino se toma y se deja aparte. Se inicia entonces una nueva iteración donde se repite el mismo proceso, pero se donde se encontrarán algunos cambios.

En el caso anterior, cuando se halló la heutística se tomó la feromona como 1, pero en esta nueva iteración se evidencia que la feromona ha cambiado en algunos caminos, lo que sucederá entonces, es que las hormigas empezarán a ignorar poco a poco los caminos con menos feromona y a frecuentar más los caminos con mayor feromona. En cada caso, la feromona aumenta o disminuye según se efectúe la operación correspondiente, esta tendrá un impacto en la nueva probabilidad de seleccionar dicho camino. La tabla 5.10 muestra lo anterior:

Camino $A-B$ (camino que aumentó la feromona inicial)	
Heurística primera iteración	Heurística segunda iteración
$1^1 \cdot \left(\frac{1}{1}\right)^3 = 1$	$(1{,}61428)^1 \cdot \left(\frac{1}{1}\right)^3 = 1{,}61428$
Camino $A-C$ (camino que disminuyó la feromona inicial)	
Heurística primera iteración	Heurística segunda iteración
$1^1 \cdot \left(\frac{1}{10}\right)^3 = 0.001$	$(0{,}935714)^1 \cdot \left(\frac{1}{10}\right)^3 = 0.0000935714$

Cuadro 5.10: Aumento y disminución de feromona

Es evidente que la feromona tendrá un gran impacto en el transcurso del algoritmo, porque al

hallar la nueva relación proporcional con la nueva heurística y la FA, dependiendo de la cantidad de feromona actual, hará que aumente la probabilidad de tomar uno u otro camino si hay más o menos cantidad. Es entonces que, si realizamos al menos 10 iteraciones, siempre haciendo los cálculos y actualizando tanto la feromona como la heurística, encontraremos que el camino mas óptimo en este grafo será $(A \;\; B \;\; D \;\; C) = 28$.

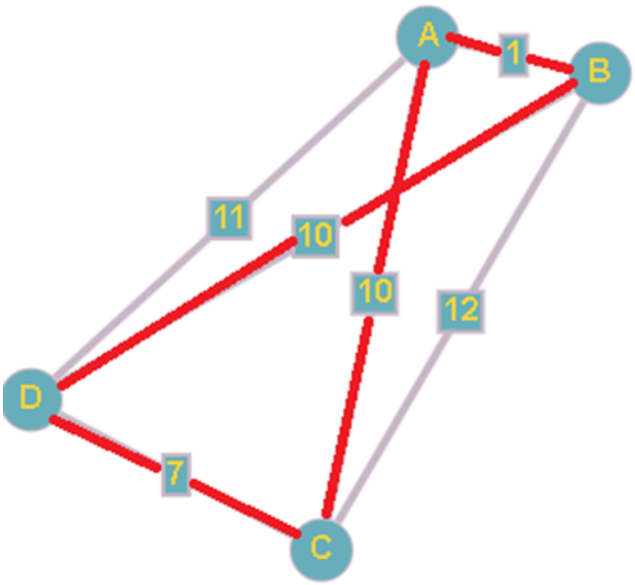


Figura 5.16: Camino Óptimo

5.4. Algoritmo de fuerza bruta

Este es uno de los algoritmos más sencillos que se han propuesto para resolver el TSP y consiste en generar todas las permutaciones posibles entre los nodos dados y compararlos entre sí para ver cuál es el camino de mínimo distancia que los conecta; el que tenga el menor valor es la solución óptima.

Sin embargo, este algoritmo es el menos aconsejable de todos debido a la demanda de recursos que requiere para su aplicación. Además, generar y evaluar todas las posibles permutaciones es inabordable en cierto punto debido a la cantidad de posibles soluciones (como se mostró en la tabla4.2.3, provocando también que el algorimto tenga el peor rendimiento de todos.

El algoritmo consiste en los siguientes pasos:

1. Calcular todos los caminos hamiltonianos.

2. Calcular los costos de cada camino hamiltoniano.

3. Seleccionar el camino con menor costo.

Su diagrama de flujo es:

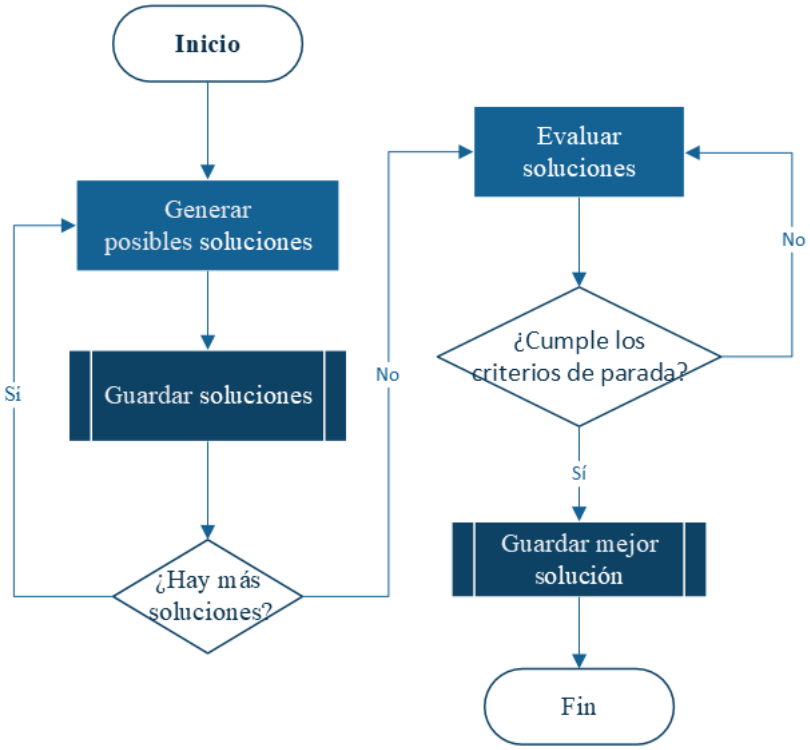


Figura 5.17: Diagrama de flujo Fuerza Bruta

5.5. Algoritmo de Held-Karp

El algoritmo Held-Karp (también llamado algoritmo Bellman-Held-Karp) es un algoritmo de programación dinámica propuesto en 1962 de forma independiente por Bellman, Held y Karp para resolver el problema del TSP. Como entrada tiene una matriz de distancia entre un conjunto de ciudades y su objetivo es encontrar el recorrido de longitud mínima que visita cada ciudad exactamente una vez antes de regresar al punto de partida. Dicho método encuentra la solución exacta a este problema y a varios problemas relacionados (incluido el problema del ciclo hamiltoniano) en tiempo exponencial.

Para el planteamiento de este método hay que tener presente que el ciclo hamiltoniano óptimo no depende del vértice que se etiqueta inicialmente, esto implica que se pueda escoger un nodo aleatoriamente y fijarlo como nodo inicial del algoritmo sin perder generalidad.

El algoritmo consiste en los siguientes pasos:

1. Se escoge un nodo inicial y se consideran los subconjuntos $S \subset \{2, ..., n\}$ de nodos faltantes. Cada

uno de los subconjuntos codifica una lista de nodos y un camino que los recorre solo una vez. Se denota la distancia entre subconjunto como $D(S, c)$.

2. Si $S = c$, entonces $D(S, c) = d_{1,c}$. En otro caso $D(S, c) = \text{mín}_{x\epsilon\ S\setminus\{c\}}\ D(S, c) = (S\setminus\{c\}, x + d_{x,c})$

3. Se escoge la distancia mínima de un ciclo Hamiltoniano que pase por todos los distintos circuitos hamiltonianos.

Y su diagrama de flujo es:

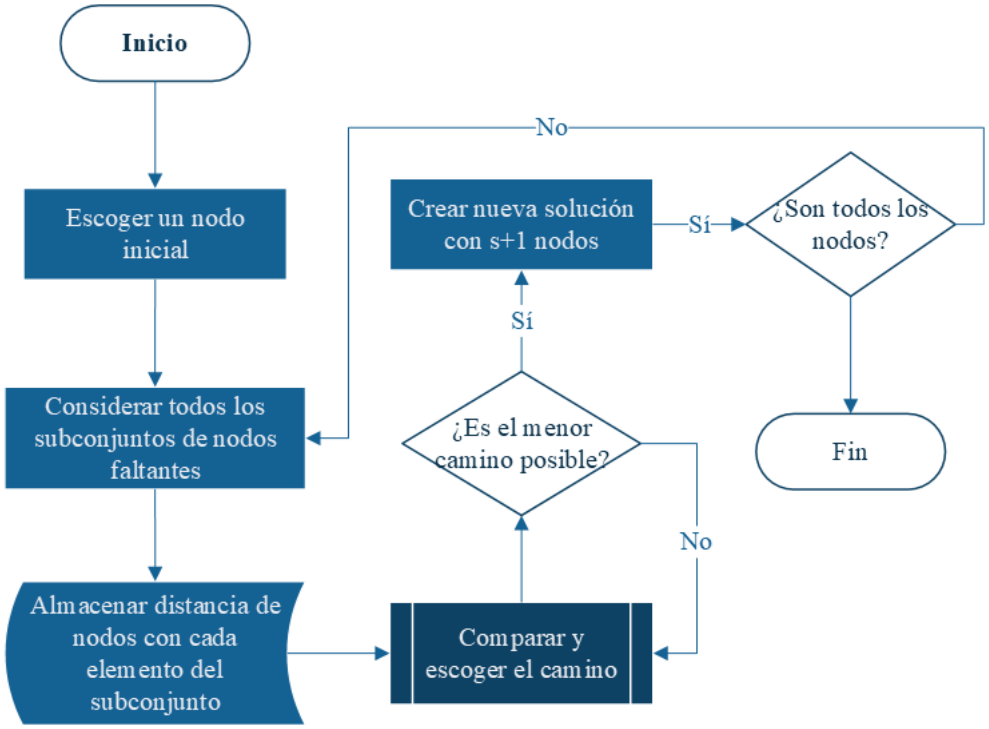


Figura 5.18: Diagrama de flujo algoritmo Held Karp

6. Resultados y análisis

En el siguiente capítulo, se hace evidencia del resultado en cinco instancias con el algoritmo creado y su comparación con otros algoritmos para solucionar el TSP. Dichas instancias son comparadas con el algoritmo Held Karp y el algoritmo de la fuerza bruta los cuales fueron mencionados anteriormente para corroborar si existe alguna correlación que nos permita afirmar si efectivamente el desarrollo propuesto es más eficiente o no. Para dicha prueba se utilizaron grafos con 5,6,7, 10 y 20 nodos generados de manera aleatoria mediante una matriz de distancias en Python.

	Algoritmo					
	ACO Creado		Held-Karp		Fuerza Bruta	
Instancia	Tiempo de ejecución	Distancia	Tiempo de ejecución	Distancia	Tiempo de ejecución	Distancia
5 nodos	0.06478238105773926	25.42106135	0.039553	25.4211	0.032553	25.4211
6 nodos	0.1155405044555664	15.466216779999998	0.037344	15.4662	0.029189	15.4662
7 nodos	0.17420744895935059	28.84059358	0.076802	27.2081	0.074312	27.2081
10 nodos	5.7197175	27.252992097769162	0.03804	27.2529921	Error	Error

Cuadro 6.1: Comparación de algoritmos

Para la instancia de 5 nodos se puede observar que el algoritmo más eficiente es el de fuerza bruta puesto que, en cuestión de distancia o coste, los tres algoritmos brindan el mismo resultado, sin embargo, la diferencia radica en el tiempo de ejecución. El algoritmo de colonia de hormigas proporciona un tiempo de ejecución aproximado de 0.064782 mientras que el de fuerza bruta 0.032553. Aquí parecería que no existe una diferencia de tiempo tan grande, sin embargo, el ACO tarda 1.99 veces más en ejecutar la misma instancia, significando una gran diferencia desde un punto de vista proporcional. A su vez, el algoritmo Held Karp es un poco más eficiente que el ACO con un tiempo de ejecución 0.039553 el cual es relativamente cercano al resultado de la fuerza bruta y representa un 61 % del tiempo que le tomaría al ACO resolver la instancia.

Por otro lado, se puede observar que la dinámica no es muy diferente en las instancias de 6 y 7 nodos, siendo el ACO el algoritmo que tarda más tiempo en ejecutarse completamente y cuya solución da

una distancia de mayor magnitud, pese a esto, el cambio significativo ocurre cuando la cantidad de nodos es mayor o igual a 10 evidenciando como el algoritmo de la fuerza bruta ya no es viable para solucionar el problema. Como se expresó durante el desarrollo del trabajo, el algoritmo de la fuerza bruta explora todas las permutaciones posibles y selecciona la que brinda una menor suma de sus aristas, es entonces, que cuando se pone en marcha el programa el mismo parece ejecutarse y se esperaba una solución del mismo, no obstante, la consola muestra un error por la sobrecarga de la memoria RAM del dispositivo, traduciendo esto a que dada la cantidad de soluciones posibles evaluar en dicha instancia la cual es de $60822550204416 \times 10^3$ el computador simplemente no cuenta con los recursos para evaluar tal cantidad de soluciones en un tiempo razonable y por lo tanto no es posible su ejecución.

Paralelamente se puede observar que para dicha instancia el algoritmo más eficiente es el Held-Karp siendo dicho algoritmo el que brinda una solución de 27.2529921 en 0.03804 segundos, a comparación de la solución en el ACO el cual es de 27.2529920 en 5.7197175 segundos, tardando 150 veces más en ejecutarse.

En cuanto a eficiencia y tiempo de ejecución parece que el algoritmo de Held-Karp es el que muestra un mejor rendimiento en general, dado que resulta ser el código más regular en las pruebas realizadas y por tal motivo la opción más eficiente para implementar, no obstante, dicha conclusión parecería caerse en la instancia de 20 ciudades, donde la cantidad de soluciones posibles debido a su comportamiento exponencial, da muestra un resultado distinto. En la instancia de 20 ciudades ,el algoritmo de la fuerza bruta brinda nuevamente un error por la cantidad de caminos posibles, el Held-Karp devuelve una solución de 40.32240 en 120.70484 segundos, como contrapartida, el ACO muestra una solución de 44.8117592 en 4.31255984 segundos, de tal forma que surge la interrogante sobre en qué contexto es indicado irse por uno u otro algoritmo, significando esto que el algoritmo Held-karp brinda una solución más óptima (en esta instancia) que el ACO, pero el ACO nos brinda una solución en poco más de 4 segundos siendo casi 28 veces más rápido en ejecutarse. Como se sabe, el añadir un nodo al problema podría implicar una subida de tiempo de ejecución impresionante por la naturaleza del problema y cada vez será más evidente la diferencia entre los tiempos de ejecución.

Por tal motivo, consideramos que el algoritmo ACO tiende a ser más eficiente en instancias de grafos con nodos muy numerosos, pues al aumentarlas brindará una solución relativamente cercana a la óptima en un tiempo bajo, en comparación con los algoritmos heurísticos utilizados, los cuales, si bien

es verdad que brindan la solución exacta del problema, no tienen una utilidad muy aprovechable si solo funcionan hasta un número limitado de nodos o se demoran demasiado tiempo en ejecutarse.

7. Conclusiones

De la indagación profunda, estudio y analisis, tanto del TSP como de los algoritmos, podemos concluir lo siguiente:

- Se realizó una indagación amplia en el estado del arte respecto al Problema del Viajante y su solución mediante el uso de algunos algoritmos exáctos, heurísticos y metaheurísticos. Por es razón, se decidió realizar la construcción de un algoritmo de colonia de hormigas en el sentido de que conceptualmente, es uno de los métodos más atractivos y fáciles para explicar a las personas que no están inmersas en un contexto tan arraigado a las ciencias de la computación u optimización matemática.

- Una de las mayores influencias en la determinación de una solución ópmita para el TSP es el tamaño que tenga, pues a medida que se comienzan a considerar más ciudades dentro del problema, la eficiencia del algoritmo para encontrar una buena solución varia significativamente con respecto a otros. En el análisis de los resultados obtenidos, al comparar el algoritmo creado con respecto a los algoritmos heurísticos notamos que en instancias pequeñas los algoritmos heuristicos pueden ser mas eficientes, sin embargo, cuando aumenta la cantidad de ciuidades estos se limitan en su uso, lo que supone la necesidad imperativa de implementar métodos metaheurísticoa si se desea una solución en cualquier instancia de cualquier tamaño.

- El ACO al ser un algoritmo hecho con el propósito de resolver el TSP, refleja las condiciones específicas necesarias para solucionar el problema y es uno de los más eficientes. Sin embargo con el ACO propuesto, al momento de mostrar las soluciones para el problema en instancias demasiado grandes, notamos que si bien no conseguimos encontrar la solución óptima en los grafos nos acercamos lo bastante en unos tiempos razonables, algo que consideramos mas útil si se aplica en un contexto de la vida real, puesto que con frecuencia priorizar una solución "efectiva"de manera rápida resulta más relevante que buscar la solución óptima.

- En la elaboración de este trabajo de grado se expusieron algunas de las diferentes metodologías que pueden implementarse para solucionar el TSP usando la programación. En ese sentido, durante el

desarrollo logramos evidenciar algunos beneficios que esta podría suponer para los estudiantes de educación básica y media entorno a ciertos procesos de la actividad matemática, teniendo en cuenta que programar no solo sirve como una herramienta práctica para aplicar conceptos matemáticos y abordar problemas de manera lógica y estructurada, sino que también fomenta un entendimiento más profundo de las situaciones y los resultados.

Así, de los cinco procesos generales necesarios para el aprendizaje de los estudiantes en matemáticas, consideramos que el razonamiento y la modelación son los que más destacan y se pueden vincular directamente con la programación. El primero, por el sentido que deberán tener los estudiantes para identificar patrones anticipando resultados y sugiriendo interpretaciones y posibles soluciones por medio de sus conocimientos previos. Y el segundo, porque la programación puede concebirse como una extensión natural de la modelización debido a la manera en que programar requiere detectar esquemas repetitivos en el mundo real para reconstruirlos mentalmente y representarlos de manera más comprensible, a través de modelos matemáticos que deban traducirse en algoritmos y ecuaciones.

- El estudio realizado en este trabajo de grado da evidencia de como la Licenciatura en Matemáticas de la Universidad Pedagógica Nacional tiene potencial para motivar al estudio de disciplinas que no suelen trabajarse a fondo por la naturaleza del pregrado. Por consiguiente, una propuesta derivada del trabajo consiste en alentar al programa a profundizar en el estudio de la programación, dado que si bien es cierto que la licenciatura incluye el estudio de esta disciplina en los primeros semestres, no siempre otorga bases lo suficientemente sólidas. Esta propuesta no solo se plantea por la creciente demanda de habilidades computacionales que el mundo laboral actual requiere, sino también por el enriquecimiento en la formación de los futuros educadores matemáticos al prorporcionarles herramientas versátiles con las cuales puedan simular, modelar y proponer actividades dentro del aula que permitan mejorar la enseñanza y personalizar el aprendizaje, demostrándoles a los estudiantes la trascendencia de las matemáticas en el mundo real.

8. Bibliografía

[1] Aldana Gonzá'les, A. (2015). Algoritmos genéticos vs enjambres de partículas: búsqueda de parámetros en una dinámica compleja.

[2] Alfaro, L. (2020). Ciclo hamiltoniano óptimo en un grafo (problema del viajante).

[3] Algarín, C. A. R. (2010). Optimización por colonia de hormigas: aplicaciones y tendencias. *Ingeniería solidaria*, 6(10-11):83–89.

[4] Alonso, S., Cordón, O., Fernández, I., and Herrera, F. (2004). La metaheurística de optimización basada en colonias de hormigas: modelos y nuevos enfoques. *Optimización inteligente: técnicas de inteligencia computacional para optimización*, pages 261–314.

[5] Baquela, E. and Redchuk, A. (2013). *Optimización Matemática con R. Volumen I: Introducción al modelado y resolución de problemas*. Bubok Publishing SL.

[6] Brucato, C. (2013). The traveling salesman problem.

[7] Crehuet Lucas, I. (2022). El problema del viajante con grafos.

[8] Dorigo, M., Birattari, and Stutzle, T. (2004). *Ant colony optimization*. The MIT Press.

[9] Dorzán, M., Gagliardi, E., Leguizamón, M., Taranilla, M., and G, H. (2009). Algoritmos aco aplicados a problemas geométricos de optimización. *XIII Encuentros de Geometrıa Computacional*.

[10] Estevez Valencia, P. (2023). Optimización mediante algoritmos genéticos. pages 83–92.

[11] García Travieso, M. V. (2014). Problema del viajante de comercio (tsp): métodos exactos de resolución.

[12] Goldberg, D. (1989). *Genetic Algorithms in search,Optimization and Machine learning*. Addison-Wesley.

[13] Guerra, J., Soberanes, H., Rodriguez, M., Valadez, J., and Magallanes, U. (2015). Análisis comparativos de metaheurísticas aplicadas al problema del tsp. *XII encuentro de Participación de la Mujer en La Ciencia*.

[14] Herrera, F. (2017). Introducción a los algoritmos metaheurísticos.

[15] Hincapié, R. A., Porras, C. A. R., and Gallego, R. A. (2004). Técnicas heurísticas aplicadas al problema del cartero viajante (tsp). *Scientia et technica*, 10(24):1–6.

[16] Holstein, D. (1998). *Una Metaheuristica Co-evolutiva para el problema del viajante de Comercio*. PhD thesis, Universidad Nacional de La Plata.

[17] Infantes Durán, M. (2018). El problema del viajante (tsp).

[18] Lamos Diaz, H., Galvan Nuñez, S. A., Gonzalez Villamizar, L. J., and Cruz Jimenez, C. (2013). Algoritmo pso-híbrido para solucionar el problema de ruteo de vehículos con entrega y recolección simultáneas. *Revista Facultad de Ingeniería*, 22(35):75–90.

[19] Martínez-Cava, C. S. (2021). El problema del viajante, heurísticas basadas en algoritmos genéticos.

[20] Mendoza Casanova, J. J. (2017). *TRAVELING SALESMAN PROBLEM (TSP) Diseño de Algoritmos Heurísticos y Metaheurísticos eficientes para resolver el Problema del Agente Viajero*. PhD thesis, Universidad Nacional Autónoma de Nicaragua.

[21] Miralles Insa, C. J. (2021). Grafos: Camino mínimo con algoritmo de bellman. Recurso en línea.

[22] Montoya Torres, L. (2020). Una aplicación del problema del viajante de comercio a la distribución del dinero en efectivo en la región de murcia.

[23] Obando-Vidal, F., Díaz-Mariño, N., and Martínez-Flor, E. (2020). Algoritmo de optimización de colonia de hormigas aplicado a tsp, una revisión sistemática. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E38):404–417.

[24] Rodríguez Castañeda, L. (2005). Algoritmos para calcular la ruta más corta en la malla vial de la ciudad de bogotá.

[25] Saiyed, A. R. (2012). The traveling salesman problem. *Indiana State University*, 2:1–15.

[26] Yang, X.-S. (2010). *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons.

[27] Yungán Cazar, J. C., Salazar Álvarez, E. G., and Villacrés Sampedro, J. E. (2022). Algoritmo de bellman ford para solucionar el problema de la ruta más corta entre nodos. *Polo del Conocimiento*, 7(7):1288–1302.