Comparación de algoritmos de optimización por enjambre de partículas y colonia de hormigas para indicar el mejor método para optimizar una red de sensores inalámbricos

Por:

David Leonardo Cifuentes Acosta

Monografía presentada como opción de grado para optar al título de Ingeniero electrónico

Aprobado por:

Ing. Sindy Paola Amaya

Director

FACULTAD DE INGENIERÍA ELECTRÓNICA

UNIVERSIDAD SANTO TOMÁS

BOGOTA D.C.

2021

Tabla De Contenido

Planteamiento del problema **4**

Justificación **5**

Estado del arte **6**

Antecedentes 6

An Improved Routing Algorithm Based on Ant Colony Optimization in Wireless Sensor Networks 6

Particle Swarm Optimization in Wireless-Sensor Networks: A Brief Survey 6

An Improved Method of WSN Coverage Based on Enhanced PSO Algorithm 6

Data Transmission Optimal Routing in WSN Using Ant Colony Algorithm 7

The Combination of Ant Colony Optimization (ACO) and Tabu Search (TS) Algorithm to Solve the Traveling Salesman Problem (TSP) 7

Improved Enhanced Self-Tentative PSO Algorithm for TSP 8

Objetivos **9**

Objetivo general 9

Objetivos específicos 9

Marco Teórico **10**

Redes de sensores inalámbricas (WSN) 10

Algoritmo de optimización por colonia de hormigas 11

Optimización por enjambre de partículas (PSO) 12

Problema del vendedor viajero 14

Metodología **15**

El algoritmo de colonia de hormigas 16

Algoritmo de optimización por enjambre de partículas 19

Análisis de resultados **23**

TSP con PSO 23

TSP con colonia de hormigas 26

Algoritmo de colonia de hormigas aplicado a la wsn planteada 27

Conclusiones **29**

Bibliografía **30**

1. Planteamiento del problema

Las redes de sensores inalámbricos o por sus siglas en inglés WSN, son redes compuestas por un gran número de nodos los cuales realizan las tareas de recolección de datos, procesamiento de datos y comunicación de datos. Estas redes son utilizadas para controlar diversas condiciones en distintos puntos, entre ellas la temperatura, el sonido, la vibración, la presión y movimiento o los contaminantes. Los sensores pueden ser fijos o móviles. Uno de los principales problemas de las WSN es el gasto de energía que tienen las redes debido a que estas redes se alimentan por medio de baterías las cuales tienen una cantidad limitada de energía, esto sumado a que cuando cada nodo transmite información consume parte de esa energía por lo que al momento de transmitir una gran cantidad de datos la energía se consume a una mayor velocidad por lo que el ciclo de vida de la red se acorta. En el siguiente documento se presentan y comparan dos algoritmos para mejorar el consumo de energía de la red, por medio de la optimización de la ruta que recorren los datos a través de la red[1].

El algoritmo de colonia de hormigas es un modelo inspirado en el comportamiento de colonias de hormigas reales. Estudios realizados explican cómo animales casi ciegos, como son las hormigas, son capaces de seguir la ruta más corta en su camino de ida y vuelta entre la colonia y una fuente de abastecimiento, esto se debe a que las hormigas pueden transmitir información entre ellas por medio de feromonas[2]. La optimización por enjambre de partículas es una opción bastante popular cuando se trata de problemas multidimensionales posee fortalezas como fácil implementación, eficiencia computacional y es rápido al momento de converger, otra característica de este algoritmo es que no es determinístico lo que quiere decir que los resultados obtenidos no serán los mismos aunque sea la misma función[3].

Con esto se busca simular una red de sensores inalámbricos e implementar una de las dos propuestas de algoritmos, para posteriormente compararlas y determinar cuál es la más eficiente para optimizar la red planteada. Esto con el objetivo de responder la siguiente pregunta. ¿Entre los algoritmos de optimización por enjambre de partículas y colonia de hormigas es más efectivo optimizar una red de sensores inalámbricos?

2. Justificación

Los algoritmos de colonia de hormigas(ACO) y enjambre de partículas(PSO), son basados en clustering, donde se implementan comportamientos de enjambre, se diferencian en que el ACO se acopla mejor a problemas donde los puntos de salida y llegada están definidos y el PSO es multiobjetivo[4]. El objetivo de este documento es aplicar uno de los dos algoritmos a una WSN simulada para optimizar tanto la transmisión de los datos como la energía que se gasta en cada nodo de esta red. Para posteriormente poder comparar los resultados de cada optimización y poder decidir cuál de los dos algoritmos se adapta mejor al problema presentado.

A medida que avanza el tiempo la sociedad está requiriendo obtener información en cualquier sitio y los más rápido posible, cada vez la demanda se volverá más grande, por lo que estas redes de sensores se deben volver más efectivas al momento de transmitir información en tiempo real así lograr mejorar la transmisión de cualquier tipo de información para cualquier entidad en el mundo.

Con esto se busca encontrar el mejor algoritmo para resolver algunos problemas que presentan las WSN y para mejorar la transmisión de datos por medio de estas redes ya que en la actualidad se están convirtiendo en una parte fundamental de la sociedad, donde las personas buscan siempre estar conectados en cualquier lugar que se encuentren y no solo las personas, las instituciones siempre manejan grandes cantidades y a diaron mueven una gran cantidad de datos por medio de WSN, por lo que hacer que sean más eficientes logrará mejorar la forma en la que se mueve la información.

3. Estado del arte

3.1. Antecedentes

3.1.1. An Improved Routing Algorithm Based on Ant Colony Optimization in Wireless Sensor Networks(2017

En este documento se busca la forma de encontrar el camino óptimo para hacer más eficiente la energía al momento de hacer el enrutamiento de datos para las WSN.

Para el modelo de la WSN en este trabajo usaron N número de sensores N distribuidos dentro de una región cuadrada de tamaño *MxM,* El nodo transmite B bits al nodo con una distancia d. Cuando la distancia d entre el envío nodo y nodo receptor es menor que d0, el modelo de espacio libre es utilizado, y la atenuación de la potencia de transmisión es inversamente proporcional al cuadrado de d. Cuando la distancia de transmisión es mayor que d0, se utiliza el modelo de desvanecimiento de rutas múltiples, y la potencia recibida cae inversamente con la cuarta potencia de d. El consumo de energía se compone de transmisión, pérdidas de circuito y pérdidas de amplificación de potencia [5].

3.1.2. Particle Swarm Optimization in Wireless-Sensor Networks: A Brief Survey(2011

En este documento se resuelven los problemas consumo energético de las WSN, la red usada para este problema agrupa los nodos en clusters, cada cluster tiene un nodo que se encarga de ser el cluster principal, los nodos de cada grupo transmiten la información al cluster principal, el cual envía los datos a una estación base este método agrupación y tener un cluster principal influye en el rendimiento y la longevidad de la red. Las soluciones de PSO se calculan de forma centralizada en la estación base para determinar las posiciones de los sensores, nodos móviles o estaciones base. la localización de los nodos se hace en dos fases En la fase de alcance, los nodos estiman sus distancias a las antenas utilizando el tiempo de propagación de la señal o la intensidad de la señal recibida y En la fase de estimación, la posición de los nodos objetivo se estima utilizando la información de alcance, ya sea resolviendo ecuaciones simultáneas o mediante un algoritmo de optimización que minimiza el error de localización error [6].

3.1.3. An Improved Method of WSN Coverage Based on Enhanced PSO Algorithm(2019

El objetivo del autor es mejorar la cobertura máxima de la red y evitar eficazmente el PSO estándar. debido a que este cae en un problema óptimo local en la última etapa de la búsqueda. Para lograr esto se encarga de mejorar el algoritmo de enjambre de partículas debido a que la velocidad de convergencia de este es lenta, esto lo hacen por medio de un ajuste del coeficiente de inercia para mejorar el PSO estándar.

Al final del experimento se demostró que la cobertura de la red de sensores inalámbricos aplicando el algoritmo mejorado subió en un 2% con respecto al algoritmo estándar demostrando que al añadir el coeficiente de inercia mejora el rendimiento de búsqueda del algoritmo[7].

3.1.4. Data Transmission Optimal Routing in WSN Using Ant Colony Algorithm(2012

En este documento comparan el algoritmo de colonia de hormigas con uno modificado debido a que el algoritmo básico tiene una baja complejidad de tiempo para resolver el problema de consumo de energía en las WSN, usando hormigas de élite logrando disminuir el tiempo en el que la colonia encuentra la ruta óptima[8].

3.1.5. The Combination of Ant Colony Optimization (ACO) and Tabu Search (TS) Algorithm to Solve the Traveling Salesman Problem (TSP) (2019

Los autores de este documento buscaban comparar un algoritmo híbrido de colonia de hormigas con búsqueda tabú para resolver el TSP y compararlo con el algoritmo normal de colonia de hormigas, el algoritmo de búsqueda tabú consiste en realizar una búsqueda local sin que esta vuelva a pasar por el mismo espacio de solución previamente recorrido[9].

Al combinar los dos algoritmos se busca encontrar las mejores rutas y tener un tiempo de ejecución más óptimo, basado en esto crearon el siguiente diagrama de flujo para el algoritmo combinado.

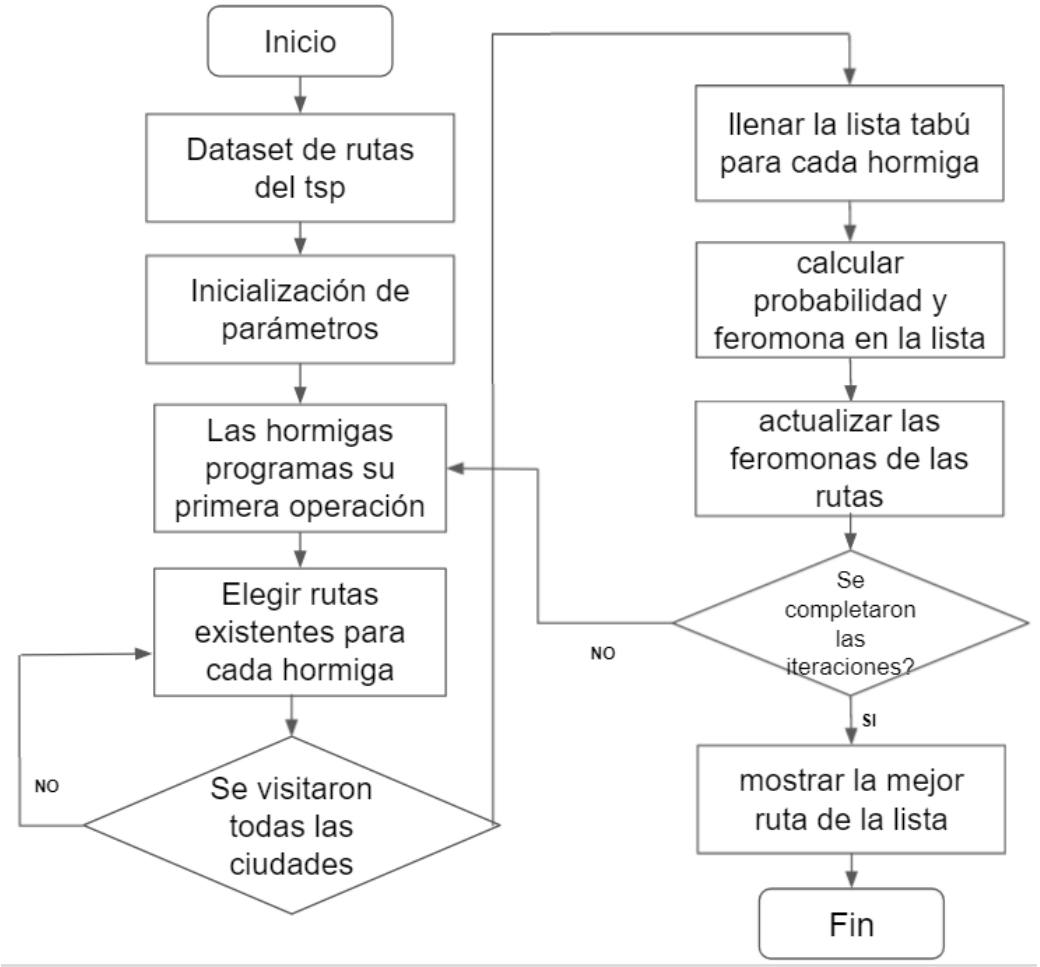

Imagen #1: Diagrama de flujo ACO-TS Propuesto por los autores de "The Combination of Ant Colony Optimization (ACO) and Tabu Search (TS) Algorithm to Solve the Traveling Salesman Problem (TSP)"

En la comparativa de los algoritmos se demostró que el algoritmo combinado encontró las mejores rutas y tuvo los mejores tiempos de ejecución que el algoritmo de colonia de hormigas normal.

3.1.6. Improved Enhanced Self-Tentative PSO Algorithm for TSP(2010

El autor propone una versión mejorada y auto-tentativa del algoritmo de enjambre de partículas para resolver el tsp, el método mejorado puede encontrar la solucion mas facil aumentando las posibilidades de encontrar la mejor solución para el problema, el autor realizó un análisis de tiempos basándose en los parámetros al momento de resolver distintas formas del tsp con un mayor número de ciudades [10].

4. Objetivos

4.1. Objetivo general

Implementar y comparar los algoritmos de optimización de colonia de hormigas y enjambre de partículas para identificar el que se adapte mejor para resolver una red de sensores inalámbricos.

4.2. Objetivos específicos

- Simular una red de sensores inalámbricos.
- Realizar una comparación entre los algoritmos de optimización de colonia de hormigas y enjambre de partículas
- Identificar el algoritmo con mejor desempeño y aplicarlo a la red de sensores propuesta

5. Marco Teórico

5.1. Redes de sensores inalámbricas (WSN)

Los inicios de estas redes, fue el proyecto Sound Surveillance System (SOSUS), el cual consiste en una cadena de puestos de escucha submarinos que se reparten en una línea que va desde Groenlandia hasta el Reino Unido pasando por Islandia. Este era usado por la armada de estados unidos para detectar submarinos soviéticos por medio de la monitorización de sonidos de baja frecuencia, el proyecto es considerado por muchos como la primera red de sensores inalámbricos [11].

Las wsn son un conjunto de sensores que están interconectados, los cuales pueden transmitir cualquier tipo de datos que se puedan monitorear por ejemplo temperatura, humedad, presión, dirección y velocidad del viento, intensidad de iluminación, intensidad de vibración, intensidad del sonido, voltaje de la línea eléctrica, concentraciones químicas, niveles de contaminantes y funciones vitales del cuerpo. Esto lo hace por medio de uno o varios canales de información utilizando tecnologías y protocolos inalámbricos como el bluetooth y wifi.

La arquitectura de las WSN está formada por un conjunto de sensores(nodos) con capacidad limitada de cómputo y comunicación, los cuales tienen su tiempo de vida está ligado a una batería, los modos se ubican de forma dispersa en una área que se puede monitorear, la información se transforma en digital en el propio nodo y transmitirla fuera de la red de sensores vía un elemento gateway a una estación base donde se almacena la información.

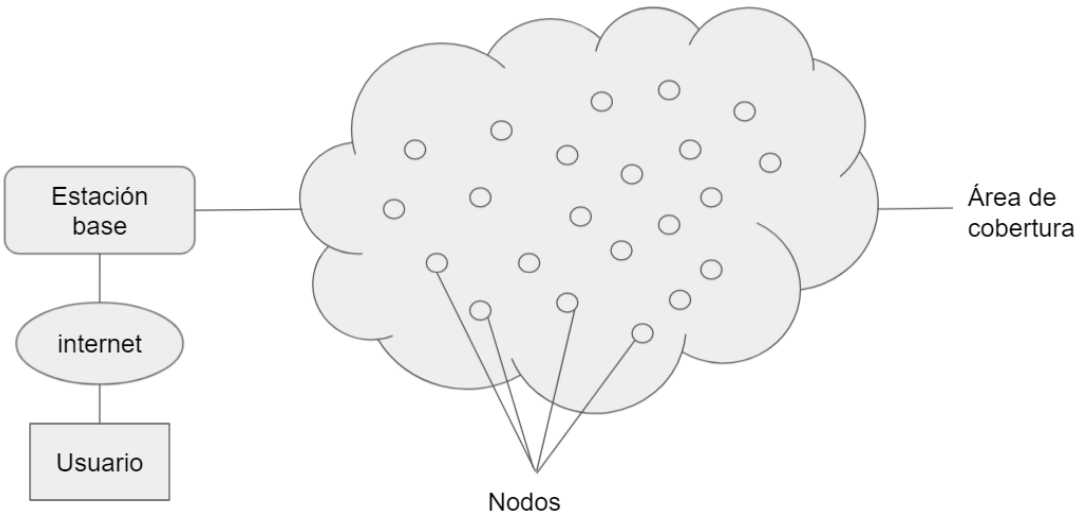

Imagen #2: Estructura básica wsn

Para alimentar estas redes normalmente se utilizan baterías las cuales son difíciles de sustituir, también se hace uso de transformadores con la salida adaptada para el nodo esto solo si se dispone de una red eléctrica. Donde la energía es consumida por los sensores, la comunicación y la transmisión de los datos, *"Por ejemplo el coste de transmisión de 1 Kb. a una distancia de 100 metros es aproximadamente el mismo que ejecutar 3 millones de instrucciones por un procesador de 100 millones de instrucciones por segundo"[12]*

5.2. Algoritmo de optimización por colonia de hormigas (ACO)

Tanto la naturaleza como el reino animal han sido de inspiración para muchas técnicas de inteligencia artificial que han surgido a través de los años, en este caso se tomó de referencia el comportamiento de las hormigas al momento que estas salen del hormiguero a conseguir comida van dejando un rastro de feromonas por el camino que recorren, para que las otras hormigas puedan seguir ese camino. El algoritmo utiliza varios agentes y rastro artificiales para simular el comportamiento de las hormigas donde los caminos más cortos le dan un valor más elevado a la feromona[13].

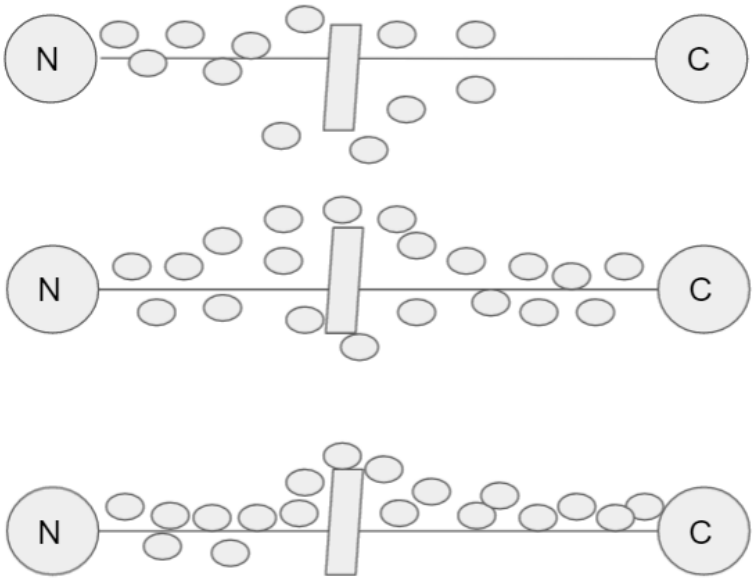

Imagen #3: Comportamiento de las hormigas

Los pasos que se siguen para este algoritmo son los siguientes.

- Una hormiga (exploradora) se mueve de manera aleatoria alrededor de la colonia.
- Si esta encuentra una fuente de comida, retorna a la colonia de manera más o menos directa, dejando tras sí un rastro de feromonas.
- Estas feromonas son atractivas, las hormigas más cercanas se verán atraídas por ellas y seguirán su pista de manera más o menos directa (lo que quiere decir que a veces pueden dejar el rastro), que les lleva a la fuente de comida encontrada por la exploradora.
- Al regresar a la colonia con alimentos estas hormigas depositan más feromonas, por lo que fortalezcan las rutas de conexión.
- Si existen dos rutas para llegar a la misma fuente de alimentos, en una misma cantidad de tiempo, la ruta más corta será recorrida por más hormigas que la ruta más larga.
- En consecuencia, la ruta más corta aumentará en mayor proporción la cantidad de feromonas depositadas y será más atractiva para las siguientes hormigas.
- La ruta más larga irá desapareciendo debido a que las feromonas son volátiles (evaporación).
- Finalmente, todas las hormigas habrán determinado y escogido el camino más corto.
- De esta forma, aunque una hormiga aislada (exploradora) se mueva esencialmente al azar, un grupo de ellas que pertenecen al mismo hormiguero decidirán sus movimientos considerando seguir con mayor frecuencia el camino con mayor cantidad de feromonas.

Una de las aplicaciones más comunes para este algoritmo es el problema del vendedor viajero, donde se tiene un número N de ciudades, cada ciudad tiene su distancia con respecto a la otra, pueden observarse en un espacio de 2 dimensiones. En esta solución las hormigas van construyendo soluciones al problema del vendedor viajero moviéndose por el grafo de una ciudad a otra hasta que completan un ciclo. Durante cada iteración del algoritmo, cada hormiga construye su recorrido ejecutando una regla de transición probabilística que indica qué nodo debe añadir al ciclo que está construyendo, donde el número de iteraciones depende del usuario[13].

5.3. Optimización por enjambre de partículas (PSO)

Igual que en el algoritmo de colonia de hormigas el PSO imita el comportamiento modelo de las actividades sociales de insectos, pájaros y peces. donde se basa en la comunicación del grupo donde los grupos de estos animales se desplazan, migran o cazan, cuando un miembro del grupo encuentra un mejor camino inmediatamente el resto del grupo sigue a ese individuo [14].

Imagen #4: Enjambre sardinas

De esta forma el PSO utiliza una población con partículas inicializadas aleatoriamente posicionadas aleatoriamente dentro de un espacio de búsqueda, cada partícula se desplaza a través del espacio de búsqueda y recuerda la mejor posición que ha encontrado cada partícula informa la mejor posición que encuentra y el resto se ajusta dinámicamente su posición y su velocidad con base a esas mejores posiciones el procedimiento del PSO se puede resumir en 6 pasos.

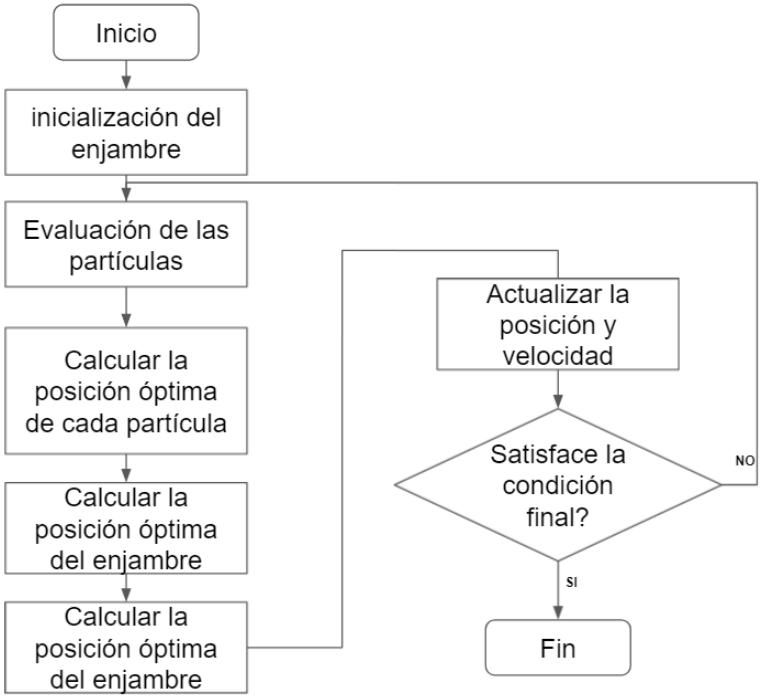

Imagen #5: Diagrama de flujo del PSO [5]

Las ecuaciones utilizadas para realizar la optimización por enjambre de partículas son las siguientes. Para calcular la velocidad de la partícula tenemos la siguiente ecuación

$$v_x = v_x + (2 * rand * (pbestx - x) + (2 * rand * (gl \quad (1)$$

Suponemos un enjambre de tamaño N, el vector de posición de cada partícula es x_i, un vector de velocidad v_i, la posición óptima de la partícula con la que se ha experimentado es P_i, la posición óptima del enjambre es P_g, por lo que la ecuación de actualización de la posición óptima del individuo es:

$$P^d_{i,t+1} = x^d_{i,t+1}, SI\ f(x_{i,t+1}) < f(P_{i,t}) \quad (2)$$

$$P^d_{i,t},\ de\ lo\ contrario$$

basado en lo anterior las ecuaciones para la velocidad y posición del enjambre quedan de la siguiente manera.

$$v^d_{i,t+1} = v^d_{i,t} + (c_1 * rand * (P^d_{i,t} - x^d_{i,t}) + (c_2 * rand * (P^d_{g,t} \quad (3)$$

$$x^d_{i,t+1} = x^d_{i,t} + v^d_{i,t+1} \quad (4)$$

Shi y Eberhart 1998 presentaron una modificación al algoritmo PSO donde en la ecuación de velocidad se tuvo en cuenta la inercia en base a esto surge una modificación a la velocidad.

$$v^d_{i,t+1} = w * v^d_{i,t} + c_1 * rand * (P^d_{i,t} - x^d_{i,t}) + c_2 * rand * (P^d_{g,t} \quad (5)$$

La siguiente ecuación es una versión del PSO donde cada partícula encuentra su propia posición óptima pbest y adicional a eso encuentra las posiciones óptimas de sus vecinos cercanos nbest esta se llama la versión local del algoritmo. mientras que en la ecuación (5) que es de la versión global, hay dos extremos la posición óptima pbest y y la posición óptima del enjambre la cual es el gbest, donde las partículas se encargan de encontrar estas dos posiciones.

$$v_{i,t+1}^{d} = w * v_{i,t}^{d} + c_1 * rand * (P_{i,t}^{d} - x_{i,t}^{d}) + c_2 * rand * (P_{l,t}^{d} - \quad (6)$$

5.4. Problema del vendedor viajero

El problema del vendedor viajero o por sus siglas en inglés TSP, es de los problemas de optimización más estudiados y utilizados para pruebas de métodos de optimización[15].

Consiste en un vendedor y un grupo de ciudades, el vendedor debe iniciar en cualquiera de la ciudades, pasar por todas la ciudades del grafo y volver al punto de partida, cada camino entre ciudades presenta un costo. El reto del problema es realizar el recorrido por todas las ciudades obteniendo el costo mínimo para el viaje[16].

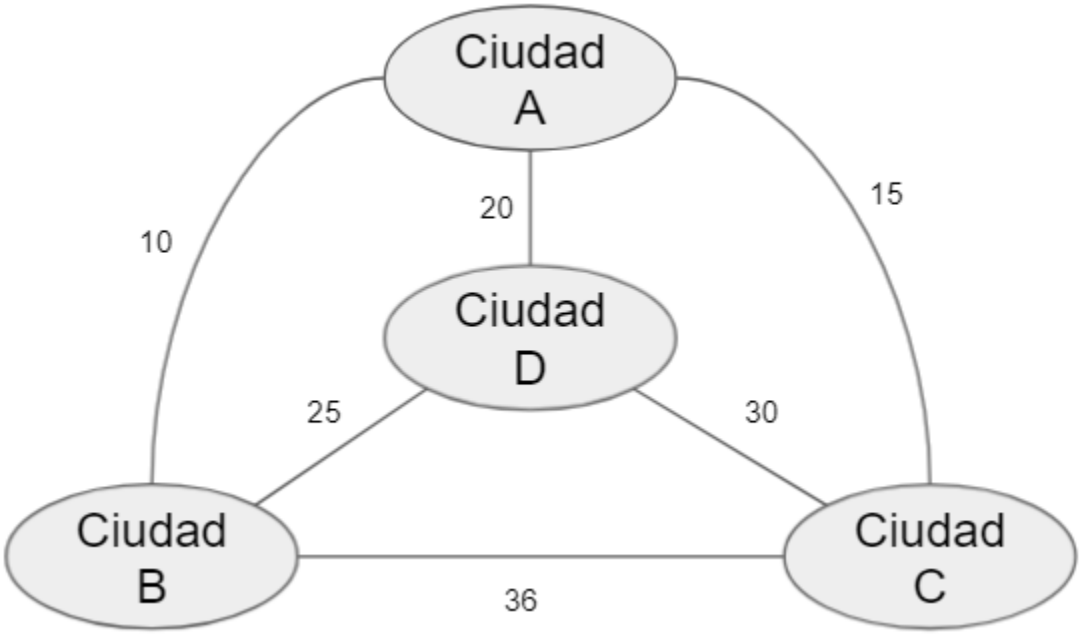

Imagen #6: Ejemplo TSP

6. Metodología

Para este experimento se realiza una metodología con escenarios simulados, donde se probaran los algoritmos propuestos (Optimización por enjambre de partículas y Algoritmo de optimización por colonia de hormigas) y se observan los resultados obtenidos, la investigación que se realizará es de tipo cuantitativa y comparativa ya que la idea es definir cuál de dos los algoritmos es mejor para resolver el problema propuesto.

- Primer paso

Para simular nuestra red de sensores inalámbricos se tomó como ejemplo los laboratorios etms de la Universidad Santo Tomás, donde cada mesa en los salones representara a un nodo dentro de nuestra red se ubican los nodos dentro de un plano. Para lograr esto se crea una lista de coordenadas las cuales son ubicadas dentro de una gráfica.

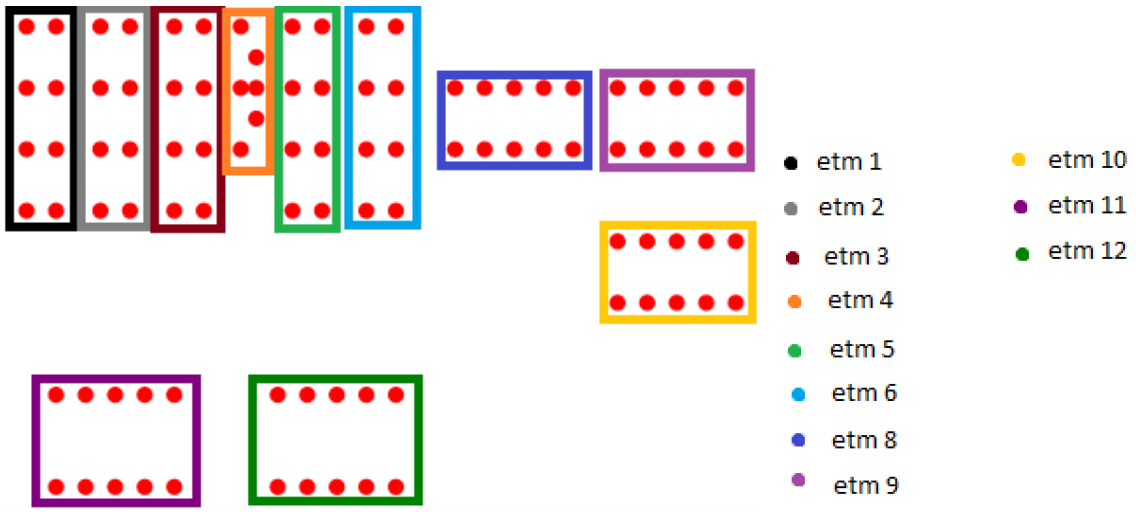

Imagen #7: WSN simulada

- Segundo paso

Se deben probar los algoritmos de optimización elegidos en este caso el algoritmo de colonia de hormigas y de enjambre de partículas, se escogió usar el problema del vendedor viajero para enfrentar los algoritmos ya que es uno de los problemas

de optimización más usados y probados, para ello se escogió un mapa con 1 ubicaciones aleatorias las cuales representan la ciudades.

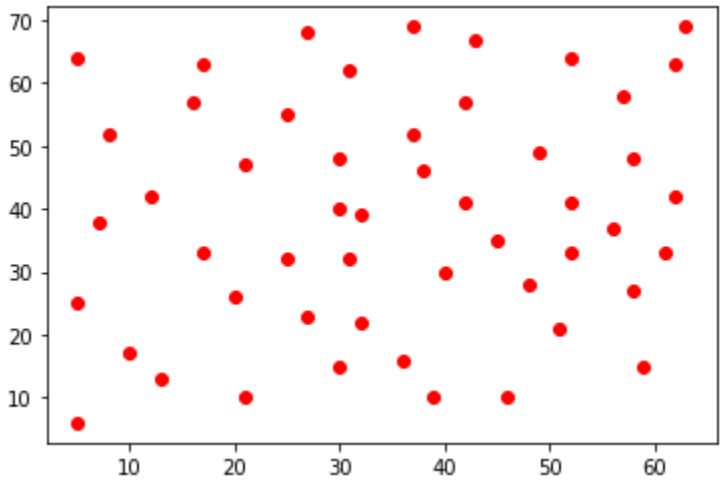

Imagen #8: Mapa de ciudades aleatorio

6.1. El algoritmo de colonia de hormigas

Se desarrolla con unos pocos pasos. Al inicio cada hormiga de la colonia crea una solución basada en los rastros de feromonas anteriores, las próximas hormigas ubicaran feromonas en los componentes de la solución elegida esto dependiendo de la calidad de la solución, cuando las hormigas terminan de crear la solución las feromonas se van evaporando, estos pasos se repiten las veces que se necesiten para generar una solución adecuada.

Para la construcción de la solución una de las hormigas seguirá el rastro de feromona más fuerte, pero para que las hormigas consideren otro tipo de solución se requiere algo de aleatoriedad en las decisiones que toman, adicionalmente también se calcula un valor heurístico y se considera que ayuda a guiar el proceso de búsqueda hacia las mejores soluciones.

El algoritmo utilizado se compone de tres clases

la clase de hormigas donde tienen

```
class Ant:
        def __init__(self, alpha, beta, num_nodes, edges):
            self.alpha = alpha
            self.beta = beta
            self.num_nodes = num_nodes
            self.edges = edges
            self.tour = None
```

```
            self.distance = 0.0

        def _select_node(self):
            roulette_wheel = 0.0
            unvisited_nodes = [node for node in
range(self.num_nodes) if node not in self.tour]
            heuristic_total = 0.0
            for unvisited_node in unvisited_nodes:
                heuristic_total +=
self.edges[self.tour[-1]][unvisited_node].weight
            for unvisited_node in unvisited_nodes:
                roulette_wheel +=
pow(self.edges[self.tour[-1]][unvisited_node].pheromone,
self.alpha) * \
                                    pow((heuristic_total /
self.edges[self.tour[-1]][unvisited_node].weight),
self.beta)
            random_value = random.uniform(0.0,
roulette_wheel)
            wheel_position = 0.0
            for unvisited_node in unvisited_nodes:
                wheel_position +=
pow(self.edges[self.tour[-1]][unvisited_node].pheromone,
self.alpha) * \
                                    pow((heuristic_total /
self.edges[self.tour[-1]][unvisited_node].weight),
self.beta)
                if wheel_position >= random_value:
                    return unvisited_node

        def find_tour(self):
            self.tour = [random.randint(0, self.num_nodes -
1)]
            while len(self.tour) < self.num_nodes:
                self.tour.append(self._select_node())
            return self.tour

        def get_distance(self):
            self.distance = 0.0
            for i in range(self.num_nodes):
                self.distance +=
self.edges[self.tour[i]][self.tour[(i + 1) %
self.num_nodes]].weight
            return self.distance
```

la clase donde se inicializan las características de los límites de las ciudades

```
class Edge:
        def __init__(self, a, b, weight,
initial_pheromone):
            self.a = a
            self.b = b
            self.weight = weight
            self.pheromone = initial_pheromone
```

la clase principal la cual contiene las dos clases anteriores.

```
class SolveTSPUsingACO:
def __init__(self, mode='ACS', colony_size=10,
elitist_weight=1.0, min_scaling_factor=0.001, alpha=1.0,
beta=3.0,
                 rho=0.1, pheromone_deposit_weight=1.0,
initial_pheromone=1.0, steps=100, nodes=None, labels=None):
        self.mode = mode
        self.colony_size = colony_size
        self.elitist_weight = elitist_weight
        self.min_scaling_factor = min_scaling_factor
        self.rho = rho
        self.pheromone_deposit_weight =
pheromone_deposit_weight
        self.steps = steps
        self.num_nodes = len(nodes)
        self.nodes = nodes
        if labels is not None:
            self.labels = labels
        else:
            self.labels = range(1, self.num_nodes + 1)
        self.edges = [[None] * self.num_nodes for _ in
range(self.num_nodes)]
        for i in range(self.num_nodes):
            for j in range(i + 1, self.num_nodes):
                self.edges[i][j] = self.edges[j][i] =
self.Edge(i, j, math.sqrt(
                    pow(self.nodes[i][0] -
self.nodes[j][0], 2.0) + pow(self.nodes[i][1] -
self.nodes[j][1], 2.0)),

initial_pheromone)
        self.ants = [self.Ant(alpha, beta, self.num_nodes,
```

```
self.edges) for _ in range(self.colony_size)]
        self.global_best_tour = None
        self.global_best_distance = float("inf")

    def _add_pheromone(self, tour, distance, weight=1.0):
        pheromone_to_add = self.pheromone_deposit_weight /
distance
        for i in range(self.num_nodes):
            self.edges[tour[i]][tour[(i + 1) %
self.num_nodes]].pheromone += weight * pheromone_to_add

    def _acs(self):
        for step in range(self.steps):
            for ant in self.ants:
                self._add_pheromone(ant.find_tour(),
ant.get_distance())
                if ant.distance <
self.global_best_distance:
                    self.global_best_tour = ant.tour
                    self.global_best_distance =
ant.distance
            for i in range(self.num_nodes):
                for j in range(i + 1, self.num_nodes):
                    self.edges[i][j].pheromone *= (1.0 -
self.rho)
```

6.2. Algoritmo de optimización por enjambre de partículas

se compone de dos clases

la primera es la clase de la partícula, en esta clase se actualizan los costos, se obtiene el costo de la ruta, actualiza los costos de cada ruta

```
class Particle:
    def __init__(self, route, cost=None):
        self.route = route
        self.pbest = route
        self.current_cost = cost if cost else
self.path_cost()
        self.pbest_cost = cost if cost else
self.path_cost()
        self.velocity = []

    def clear_velocity(self):
        self.velocity.clear()
```

```
    def update_costs_and_pbest(self):
        self.current_cost = self.path_cost()
        if self.current_cost < self.pbest_cost:
            self.pbest = self.route
            self.pbest_cost = self.current_cost

    def path_cost(self):
        return path_cost(self.route)
```

La segunda es la clase donde se realiza el PSO, esta clase genera las rutas y basados en el global best el cual encuentra la mejor solución dentro de un conjunto de soluciones basándose en las actualizaciones de posición y velocidad de las partículas.

```
class PSO:
    def __init__(self, iterations, population_size,
gbest_probability=1.0, pbest_probability=1.0, cities=None):
        self.cities = cities
        self.gbest = None
        self.gcost_iter = []
        self.iterations = iterations
        self.population_size = population_size
        self.particles = []
        self.gbest_probability = gbest_probability
        self.pbest_probability = pbest_probability

        solutions = self.initial_population()
        self.particles = [Particle(route=solution) for
solution in solutions]

    def random_route(self):
        return random.sample(self.cities, len(self.cities))

    def initial_population(self):
        random_population = [self.random_route() for _ in
range(self.population_size - 1)]
        greedy_population = [self.greedy_route(0)]
        return [*random_population, *greedy_population]
        # return [*random_population]

    def greedy_route(self, start_index):
        unvisited = self.cities[:]
        del unvisited[start_index]
        route = [self.cities[start_index]]
```

```
        while len(unvisited):
            index, nearest_city = min(enumerate(unvisited),
key=lambda item: item[1].distance(route[-1]))
            route.append(nearest_city)
            del unvisited[index]
        return route

    def run(self):
        self.gbest = min(self.particles, key=lambda p:
p.pbest_cost)
        print(f"initial cost is {self.gbest.pbest_cost}")
        plt.ion()
        plt.draw()
        for t in range(self.iterations):
            self.gbest = min(self.particles, key=lambda p:
p.pbest_cost)
            if t % 20 == 0:
                plt.figure(0)
                plt.plot(pso.gcost_iter, 'g')
                plt.ylabel('Distance')
                plt.xlabel('Generation')
                fig = plt.figure(0)
                fig.suptitle('pso iter')
                x_list, y_list = [], []
                for city in self.gbest.pbest:
                    x_list.append(city.x)
                    y_list.append(city.y)
                x_list.append(pso.gbest.pbest[0].x)
                y_list.append(pso.gbest.pbest[0].y)
                fig = plt.figure(1)
                fig.clear()
                fig.suptitle(f'pso TSP iter {t}')

                plt.plot(x_list, y_list, 'ro')
                plt.plot(x_list, y_list, 'g')
                plt.draw()
                plt.pause(.001)
            self.gcost_iter.append(self.gbest.pbest_cost)

            for particle in self.particles:
                particle.clear_velocity()
                temp_velocity = []
                gbest = self.gbest.pbest[:]
                new_route = particle.route[:]

                for i in range(len(self.cities)):
```

```
                    if new_route[i] != particle.pbest[i]:
                        swap = (i,
particle.pbest.index(new_route[i]), self.pbest_probability)
                        temp_velocity.append(swap)
                        new_route[swap[0]],
new_route[swap[1]] = \
                            new_route[swap[1]],
new_route[swap[0]]

                for i in range(len(self.cities)):
                    if new_route[i] != gbest[i]:
                        swap = (i,
gbest.index(new_route[i]), self.gbest_probability)
                        temp_velocity.append(swap)
                        gbest[swap[0]], gbest[swap[1]] =
gbest[swap[1]], gbest[swap[0]]

                particle.velocity = temp_velocity

                for swap in temp_velocity:
                    if random.random() <= swap[2]:
                        new_route[swap[0]],
new_route[swap[1]] = \
                            new_route[swap[1]],
new_route[swap[0]]

                particle.route = new_route
                particle.update_costs_and_pbest()
```

Se toman los dos algoritmos y se realizan pruebas modificando los distintos parámetros que tienen para determinar cual es más efectivo para solucionar el problema del vendedor viajero.

- Tercer paso

Analizar los resultados que se obtuvieron de ambos algoritmos con él grafo aleatorio. Basado en el desempeño de los algoritmos se tomará el mejor y se aplicará al problema presentado, modificando los parámetros para obtener el menor costo para la resolución.

7. Análisis de resultados

7.1. TSP con PSO

Para el enjambre de partículas se selecciona un dataset de ciudades con 51 ciudades la cuales son ubicadas en un grafo, donde cada camino entre ciudades tiene un costo la cual se calcula teniendo en cuenta la ubicación en el grafo de la ciudad actual y la siguiente ciudad en el recorrido.

- Primer caso
  - iteraciones: 1200
  - población de enjambre:300
  - coste inicial: 513.61
  - coste final: 498.97

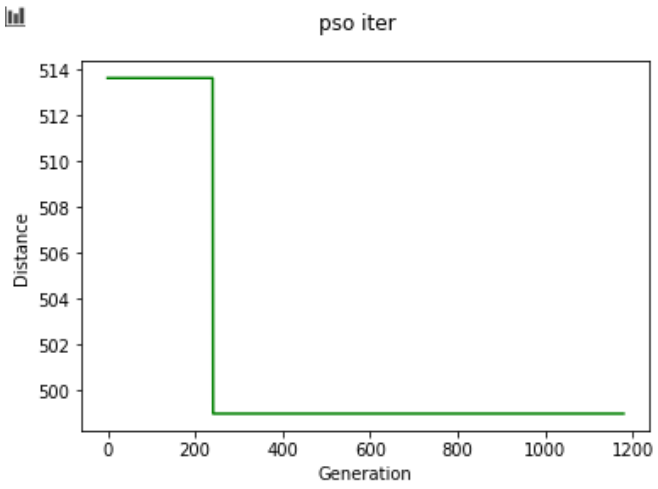

Imagen #8: Gráfica de iteraciones PSO pobl:300

Se puede observar que con una población de 300 presenta una convergencia en la iteración 240, pero vemos que el coste final no se disminuye significativamente comparado con el inicial no baja de 498.97l.

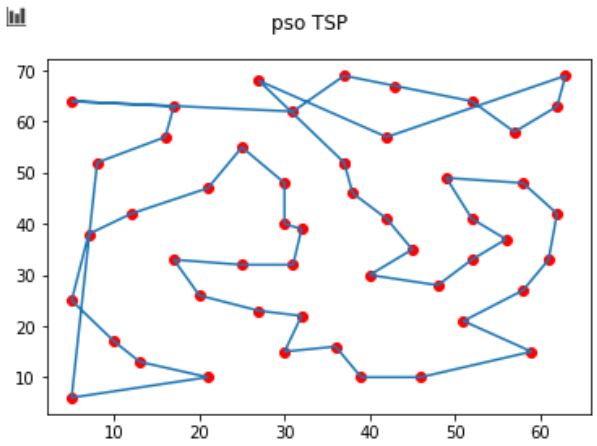

Imagen #9: Ruta final Primer caso

- Segundo caso
  - iteraciones: 1200
  - población de enjambre:600
  - coste inicial: 513.61
  - coste final: 482.56

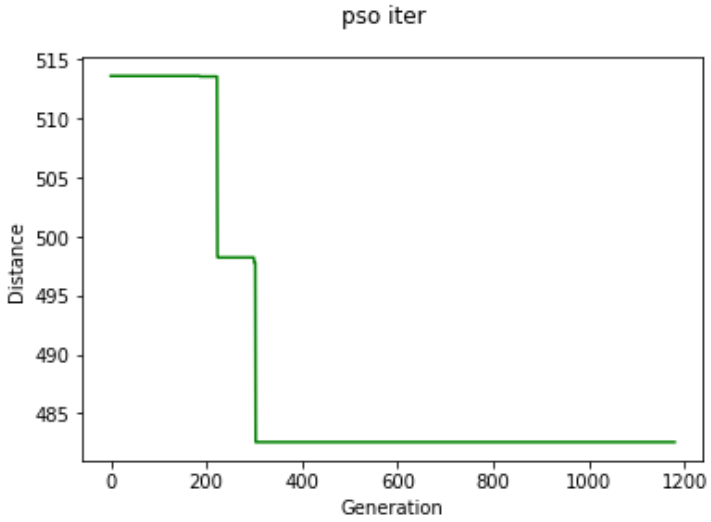

Imagen #10: Gráfica de iteraciones PSO pobl:600

Para la población de 600 se observa que empieza a converger mucho más antes desde la iteración 180, y comparado a la población de 300 el coste final si se ve que tiene una disminución bastante mayor.

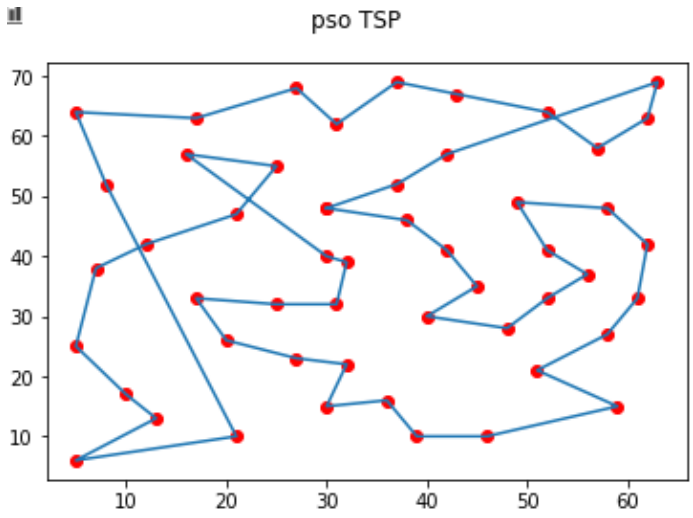

Imagen #11: Ruta final segundo caso

Igualmente se presenta una ruta distinta a la anterior.

Se fue aumentando la población pero no se observaron mayores cambios en el coste final de la ruta lo mínimo que se obtuvo fue con los siguientes parámetros.

- Tercer caso
  - iteraciones: 1200
  - población de enjambre:3200
  - coste inicial: 513.61
  - coste final: 472.27

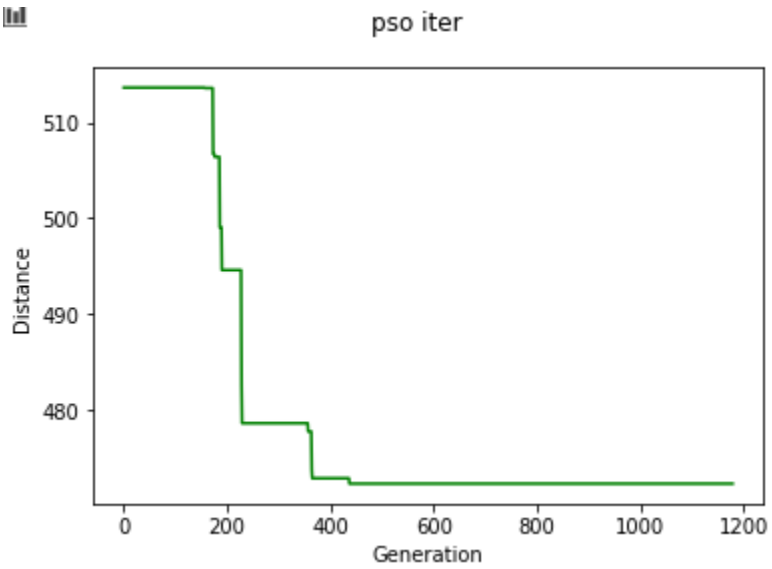

Imagen #12: Gráfica de iteraciones PSO pobl:3200

Se puede observar que al tener un mayor número de partículas, se logra converger con menos iteraciones.

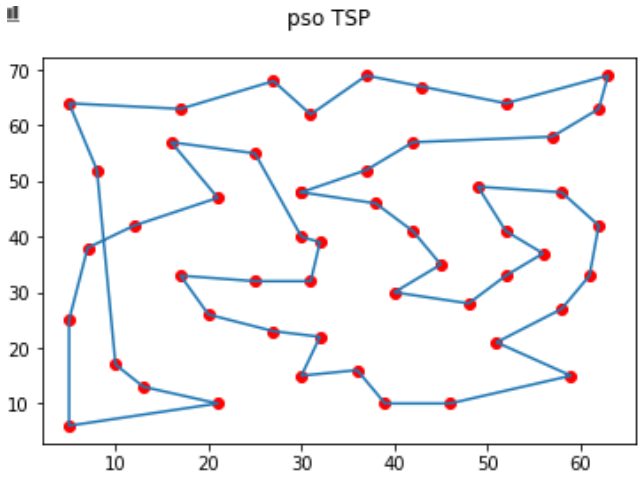

Imagen #13: Ruta final tercer caso

7.2. TSP con colonia de hormigas

Para el algoritmo de colonia de hormigas se utilizó el mismo grafo con la misma cantidad de ciudades que el PSO, también se pocuro que se realizará el mismo número de iteraciones.

- Caso #1: 5 hormigas
  - Coste: 448.48
  - iteraciones: 1200

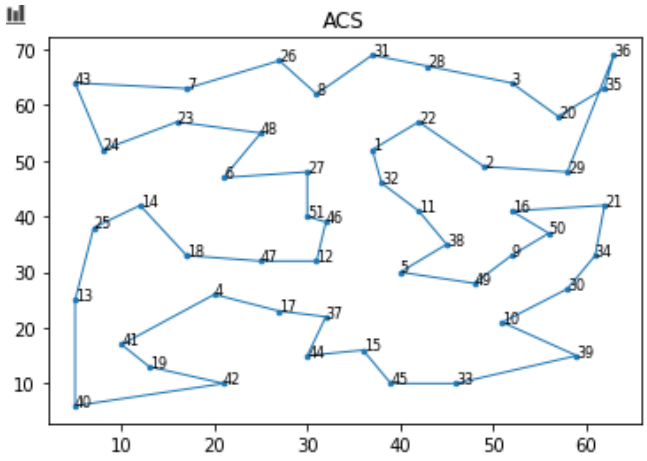

Imagen #14: Ruta final caso #1

Se logra observar que el coste con 5 hormigas ya de por si es menor al de los PSO. En cuanto a las pruebas con este algoritmo se observó que el costo no varía

mucho a medida que se aumentan las hormigas, por ejemplo con 50 hormigas se observa apenas una disminución de 5 en el costo, dado un total de 443.38 en el costo de la ruta.

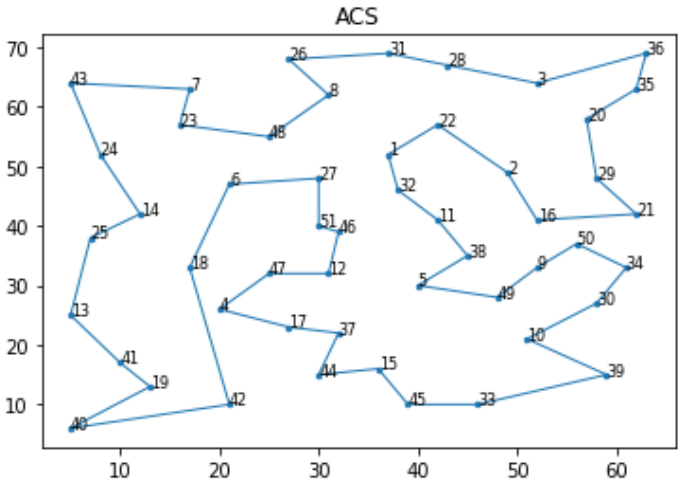

Imagen #15: Ruta final caso #2

Población PSO	costo PSO	Tiempo de ejecución (s)	Población ACO	costo ACO	Tiempo de ejecución (s)
300	498.97	42.76	5	448.48	13.26
600	482.56	56.29	50	443.38	116.97
3200	472.27	154.19	100	447.55	224.93

Tabla#1: Comparación resultados de algoritmos

Basado en las pruebas realizadas con el TSP se puede decir que el algoritmo de colonia de hormigas es una mejor opción ya que se puede conseguir un menor costo con un menor número de población, y además con la población de 5 hormigas se obtiene un menor tiempo y menor costo que el PSO.

7.3. Algoritmo de colonia de hormigas aplicado a la wsn planteada

Ahora se probará el algoritmo de colonia de hormigas con la red de los laboratorios etm que planteamos anteriormente, cuyo coste inicial es de 250.

Iteraciones	Hormigas	Coste
300	5	245.38
600	5	242.62

1200	5	240.81
300	15	240.81
600	15	233.66
1200	15	242.66

Tabla#2: pruebas del algoritmo con la WSN planteada

De acuerdo a la tabla 1 se puede observar que puede existir una equivalencia al momento de aumentar la hormigas, por lo que se puede ver en el caso de las 15 hormigas con 300 iteraciones y las 5 hormigas con 1200 iteraciones, que básicamente se obtuvo el mismo coste pero al tener menos iteraciones el tiempo de resolución es más bajo para las 15 hormigas.

Después de las 15 hormigas el algoritmo presentaba tiempos de ejecución altos por lo que se decidió dejar de aumentar más el número de hormigas.

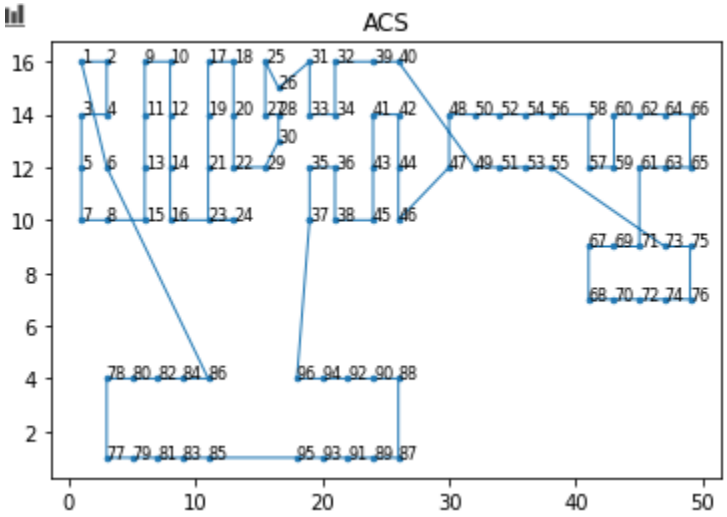

Imagen #16: Ruta con 15 hormigas y 600 iteraciones

El mapa de la red se pueden presentar ciertas variaciones, como se da el caso en el que cierren alguno de los laboratorios, por lo que no habrían ciertos nodos en la red.

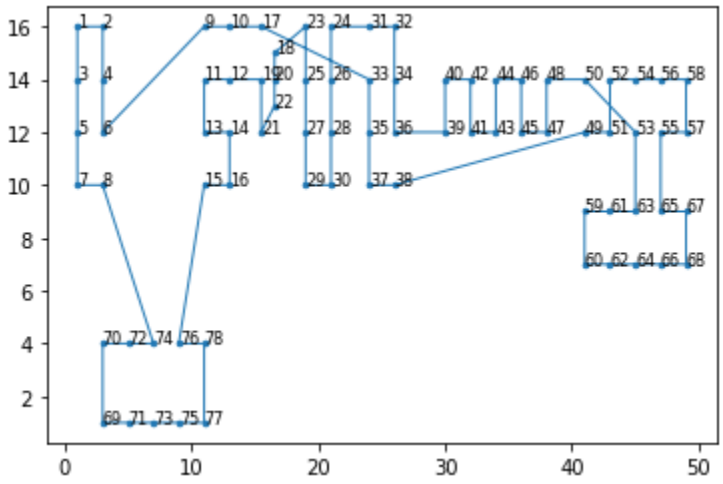

Imagen #17: Simulación con ETM 2 y ETM 12 cerrados

Iteraciones	Hormigas	Coste
300	5	214.84
600	5	204.47
1200	5	206.99
300	15	203.8
600	15	205.74
1200	15	203.16

Tabla#2: pruebas del algoritmo menor cantidad de nodos

Al existir zonas sin nodo dentro del grafo, de acuerdo a la tabla 2 se puede observar que el coste de las rutas que plantea el algoritmo se ven reducidos en una gran cantidad, esto se debe a que la red planteada posee una menor cantidad de nodos por lo que la información viaja más rápido a través de esta.

8. Conclusiones

- Debido a los resultados obtenidos el ACO resultó ser una mejor opción para resolver el problema planteado, ya que el algoritmo presenta un mejor porcentaje en cuanto al costo de la ruta se observa comparando las poblaciones más pequeñas utilizadas en los dos algoritmos, que en el caso del PSO es de 300 y del ACO es de 5, se observa una diferencia del 10% en el costo final de la ruta, en aproximadamente un tercio del tiempo de lo que lo realizó el PSO.
- Ambos algoritmos imitan un comportamiento basado en enjambres, pero el ACO es más eficiente cuando se trata de resolver problemas con un punto de partida y de llegada.
- En cuanto a tiempos de ejecución el algoritmo de enjambre de partículas presenta tiempos más bajos a comparacion de el de colonia de hormigas, ya que en las 1200 iteraciones el PSO con una población bastante grande presenta tiempos por debajo de los 100 segundos, mientras que el ACO con una población de 15 hormigas presenta tiempos por encima de ese tiempo.
- El coste computacional de estos algoritmos aumenta dependiendo en el caso del ACO la cantidad de hormigas cómo se logró observar al momento

de hacer una prueba con un número de hormigas mayor a 15, donde los recursos consumidos por el algoritmo eran bastante altos.

9. Bibliografía

[1] S. E. Campaña Bastidas, H. E. Cabrera Meza, A. J. Cervelion Bastidas, y A. Aguirre Cabrera, «Capítulo 1: Las redes de sensores inalámbricas, arquitectura y aplicaciones», *book*, may 2019.

[2] R. A. Carlos Arturo, “Optimización por colonia de hormigas: aplicaciones y tendencias”, Articulo, pp 83-89, 2010.

[3] “Optimización por enjambre de partículas”, Retrieved from https://amp.what-this-it.com/5965339/1/optimizacion-por-enjambre-de-particulas.html, 2020

[4] S. Molina, G. Leguizamon, “Algoritmos de Inteligencia de Enjambres Orientados a Map Reduce”, Articulo.

[5] Y. Sun, W. Dong and Y. Chen, "An Improved Routing Algorithm Based on Ant Colony Optimization in Wireless Sensor Networks," in IEEE Communications Letters, vol. 21, no. 6, pp. 1317-1320, June 2017, doi: 10.1109/LCOMM.2017.2672959.

[6] R. V. Kulkarni and G. K. Venayagamoorthy, "Particle Swarm Optimization in Wireless-Sensor Networks: A Brief Survey," in IEEE Transactions on Systems, Man, and

Cybernetics, Part C (Applications and Reviews), vol. 41, no. 2, pp. 262-267, March 2011, doi: 10.1109/TSMCC.2010.2054080.

[7] H. Kong and B. Yu, "An Improved Method of WSN Coverage Based on Enhanced PSO Algorithm," 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2019, pp. 1294-1297, doi: 10.1109/ITAIC.2019.8785849.

[8] Su Jun, V. Yatskiv, A. Sachenko and N. Yatskiv, "Data transmission optimal routing in WSN using ant colony algorithm," Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science, 2012, pp. 342-343.

[9] R. W. Dewantoro, P. Sihombing and Sutarman, "The Combination of Ant Colony Optimization (ACO) and Tabu Search (TS) Algorithm to Solve the Traveling Salesman Problem (TSP)," 2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM), 2019, pp. 160-164, doi: 10.1109/ELTICOM47379.2019.8943832.

[10] J. Zhang and W. Si, "Improved Enhanced Self-Tentative PSO algorithm for TSP," 2010 Sixth International Conference on Natural Computation, 2010, pp. 2638-2641, doi: 10.1109/ICNC.2010.5583011.

[11] Sound Surveillance System (SOSUS), 7(2). Retrieved from https://dosits.org/galleries/technology-gallery/locating-objects-by-listening-to-their-sounds/sound-surveillance-system-sosus/

[12] R. Fernández Martínez, J. Ordieres Meré, F.J. Martínez de Pisón Ascacíbar, A. González Marcos, F. Alba Elías, R. Lostado Lorza y A.V. Pernía Espinoza, «redes inalámbricas de sensores: teoría y aplicación práctica», Book,2009

[13] B.Barán, «Colonia de Hormigas en un Ambiente Paralelo Asíncrono», Artículo científico.

[14] D. Wang, D. Tan y L. Liu, «Particle swarm optimization algorithm: an overview», Articulo, ene 2017

[15] F.Sancho Caparrini, Algoritmos de hormigas y el problema del viajante, Retrieved from http://www.cs.us.es/~fsancho/?e=71, nov 2018

[16] “Chapter 10: The Traveling Salesman Problem ”, Retrieved from https://www.csd.uoc.gr/~hy583/papers/ch11.pdf, nov 2018