Ruta óptima para los colaboradores de la fuerza de ventas de una microfinanciera

Laura Gómez Bustamante[1], Paola Mejía Domenzain[2], Ana Bertha Coronel[3], Miguel Angel Millan Dorado[4], Marco Julio Monroy[5], and Diego Michell Villa Lizárraga[6]

[1] Alumnos de Maestría en Ciencias de Datos (ITAM)

En este trabajo se presenta la aplicación del problema del vendedor viajero (TSP por sus siglas en inglés) en la identificación de las rutas con la distancia mínima que los colaboradores de la fuerza de ventas de una microfinanciera deberían realizar para el seguimiento a sus clientes. Se presentan dos algoritmos comunes *Particule Swarm Optimization* y *Simulated Annealing*, su implementación en paralelo y una propuesta de visualización.

Traveling Salesman Problem | Particle Swarm | Simulated Annealing

I. Introducción

CM es una microfinanciera con más de 1.7 millones de clientes, que atiende principalmente a mujeres y tiene cobertura en toda la República Mexicana. Cuenta con aproximadamente 5,400 colaboradores en la fuerza de ventas que se encargan de la atracción de nuevos clientes y el seguimiento semanal en el cobro de las fichas del crédito. Así, la cartera de clientes a la que se le da seguimiento crece cada semana mediante la colocación y, los colaboradores tienen, en promedio, 298 clientes.

El área de Planeación Logística es la encargada de analizar la carga de trabajo que tienen los colaboradores de la fuerza de ventas y, actualmente se cuenta con agrupaciones de clientes por zonas sin hacer especificación alguna sobre el orden en el que se debería realizar la visita. Esto hace que, en ocasiones, la fuerza de ventas tenga que solicitar un bono adicional al ya otorgado en el bono operativo y que realice trayectos con distancias que no necesariamente son las mínimas.

La ruta de un colaborador inicia y termina en la sucursal y éste visita una sola vez a cada uno de sus clientes; es decir, la ruta del colaborador se podría describir como un ciclo hamiltoniano y, un problema que busca aquel con la distancia o costo mínimo es el problema del vendedor viajero o TSP (*Traveling Salesman Problem*). Bajo este contexto, y sin hacer distinción en el día de la semana en la que se realiza la visita, el objetivo del presente trabajo es encontrar la ruta de los colaboradores de la fuerza de ventas con la menor distancia.

Sea G(V,E) un grafo en donde V es el conjunto de nodos y E es el conjunto de aristas, se definen los siguientes campos que sirven de insumo para la construcción de los grafos:

El trabajo está dividido en ocho apartados; en el segundo, se describe brevemente el problema del vendedor viajero así como los algoritmos que se han desarrollado; en el tercer y cuarto apartados, se explican los algoritmos *Particle Swarm Optimization* y *Simulated Annealing*, respectivamente. En el quinto, se explica la propuesta de solución al problema presentado. La evaluación de los algoritmos y los resultados de la implementación en paralelo son el contenido de los apartados siete y ocho; finalmente, en el apartado 9, se ejemplifica el producto final: un *dashboard* para la visualización de las rutas.

Variable	Descripción
fza_ventas	Número de nómina del colaborador de la fuerza de ventas
no_cliente	Identificador único del cliente
lat_destino	Latitud del punto de destino
lon_destino	Longitud del punto de destino
id_origen	Identificador de la sucursal
estado	Estado en donde se encuentra la sucursal
lat_origen	Latitud del punto de origen
lon_origen	Longitud del punto de origen

Tabla 1. Diccionario de datos

II. Travelling Salesman problem

El problema del vendedor viajero (TSP por sus siglas en inglés) es uno de los problemas más estudiados en optimización combinatoria *NP-hard* que busca encontrar el ciclo hamiltoniano más corto, ya sea en distancia o en tiempo. Sea G(V,E) un grafo dirigido en donde V es el conjunto de nodos y E el conjunto de aristas, se dice que el grafo tiene un ciclo hamiltoniano si contiene un ciclo y cada nodo se visita exactamente una vez. Por ejemplo, si se visitan varias ciudades y para cada par de ciudades se calcula la distancia; se debe encontrar la ruta más corta en la que se visite una y solo una vez cada ciudad.

El problema del vendedor viajero tiene muchas aplicaciones prácticas tales como: rutas de vehículos, transmisión de datos en las redes computacionales, programación de horarios y reconocimiento de patrones.

Entre los algoritmos que se han desarrollado para tratar de resolver el TSP, se encuentran los que utilizan programación dinámica o algoritmos *branch and bound* que permiten llegar a un mínimo global y los modelos heurísticos, que a pesar de que tardan menos en la ejecución, no garantizan que se llegue a un mínimo global. *Particle Swarm Optimization* y *Simulated Annealing* son ejemplos de estos últimos ((8)) y serán desarrollados para la obtención de las rutas óptimas.

III. Particle Swarm Optimization

Descripción. El algoritmo *Particle Swarm Optimization* (PSO) se presentó por primera vez en 1995 por Eberhart y Kennedy, y fue desarrollado bajo la inspiración de las reglas de comportamiento de parvadas, bancos de peces y comunidades humanas. El PSO se implementa en poblaciones, buscando optimizar la población mediante el intercambio de información entre individuos. La solución óptima la encuentra a partir de un punto de inicio que se toma aleatoriamente del grupo de soluciones para, posteriormente buscar repetidamente sobre él.

PSO asume muy pocos supuestos sobre el problema que se está optimizando y, por ende, puede buscar soluciones en espacios grandes. Sin embargo, no garantiza que siempre se encuentre una solución óptima. Para ser más específicos, PSO no busca optimizar el gradiente del problema, es decir, no requiere que el problema sea diferenciable como es requerido por otros métodos clásicos de optimización. Por lo anterior, este algoritmo es utilizado en problemas de optimización que son parcialmente irregulares, ruidosos o que cambian constantemente a través del tiempo.

Algoritmo. Idea

- Se considera una población o enjambre ("*swarm*") de soluciones candidatas (partículas, "*particles*").
- Las partículas se mueven en un espacio de búsqueda.
- Los movimientos de las partículas son guiados por su mejor posición conocida y la mejor posición conocida del enjambre.
- Cuando mejores posiciones son descubiertas, éstas guiarán los movimientos del enjambre.
- El proceso se repite hasta encontrar satisfactoriamente una solución (No se garantiza que siempre se encontrará una solución).

Implementación

- Se busca minimizar una función de costo.
- La meta es buscar una solución a para la cual $f(a) < f(b)\ \forall\ b$ en el espacio de búsqueda, lo cual significaría que a es el mínimo global.
- Cada partícula tiene un valor *fitness* (mejor posición que ha logrado alcanzar) que es determinado por una función objetivo, la cual tiene una *velocidad* que determina su *destino* y *distancia*.
- PSO inicialmente es un grupo de partículas aleatorias (soluciones aleatorias).
- Las soluciones óptimas son encontradas por medio de búsqueda iterativa.
- Todas las partículas determinarán su siguiente movimiento considerando tanto sus mejores experiencias individualmente, como las de sus compañeros en el enjambre a través de las siguientes ecuaciones:

$$X_{id} = X_{id} + V_{id} \quad [1]$$

$$V_{id} = V_{id} \oplus \alpha(P_{id} - X_{id}) \oplus \beta(P_{gd} - X_{id}) \quad [2]$$

en donde X_{id} representa la nueva posición de la partícula y V_{id} indica la velocidad. Este último término depende de α, que representa la influencia de la mejor posición individual de cada partícula y β, que denota la influencia de la mejor posición que ha encontrado el enjambre de partículas (posición global). Por su parte P_{id} y P_{gd} indican explícitamente las mencionadas mejores posiciones de la partícula y del enjambre respectivamente.

IV. Simulated Annealing

Descripción. Al igual que PSO, el algoritmo de recocido simulado (SA, por sus siglas en inglés) tiene como objetivo general encontrar una buena aproximación al valor óptimo ("óptimo global") de una función en un espacio de búsqueda grande. Desarrollado en la década de los 80's, este algoritmo presenta una analogía con el proceso de recocido en metalurgia. Este proceso consiste en calentar a una temperatura alta el material, con la intención de propiciar mayor movilidad de sus moléculas y por ende mayor ductilidad, para después enfriarlo lentamente hasta obtener una estructura molecular estable y fuerte. En este proceso es de suma importancia el enfriamiento lento del material, de otra manera su estructura presentaría imperfecciones.

Con estas ideas, se puede resumir el algoritmo SA de la siguiente manera:

El algoritmo SA es un método de búsqueda por entornos caracterizado por un criterio de aceptación de soluciones vecinas. Éste simula los cambios de energía en un sistema sometido a un proceso de enfriamiento hasta que converja a un estado de equilibrio. La noción de enfriamiento lento implementada en el algoritmo SA, se interpreta como una lenta disminución de la probabilidad de aceptar peores soluciones mientras se explora el espacio de la solución.

Algoritmo. Idea

Se puede describir al algoritmo a través de los siguientes pasos:

- Se inicializa una solución (s) aleatoria.
- Se hace uso de una variable de temperatura (T), cuyo valor determina la medida en que pueden ser aceptadas soluciones vecinas peores que la actual.
- Se inicializa la temperatura inicial (T_0) (valor alto), la cual se reduce en cada iteración mediante un mecanismo de enfriamiento de la temperatura hasta alcanzar una temperatura final (T_f).
- Durante cada iteración se genera un número específico de vecinos ($L(t)$).
- Para cada generación de un vecino (s_v), se aplica el criterio de aceptación para decidir si reemplaza a la solución actual (s).
- Si la solución vecina es mejor que la actual, se acepta en automático.
- Por otro lado, si ésta es peor, aún existe la probabilidad de que el vecino sustituya a la solución actual. De esta forma SA evita caer en óptimos locales.
- Esta probabilidad depende de la diferencia de energía entre la solución actual y la vecina ($\Delta E = f(s_v) - f(s)$) y de la temperatura: $P_{acep} = e^{\frac{-\Delta E}{T}}$. Así, a mayor temperatura, mayor probabilidad de aceptación de soluciones peores. Por ende, al principio de la ejecución el algoritmo explora y al final puntualiza.
- Una vez finalizada la primera iteración (tras generar $L(T)$ soluciones vecinas), se enfría la temperatura y se vuelve a iterar una y otra vez hasta alcanzar el óptimo.

Observar figura 1 tomada de (**?**) , la cual muestra un pseudocódigo del algoritmo SA.

Algorithm 2.3 Template of simulated annealing algorithm.

Input: Cooling schedule.
$s = s_0$; /* Generation of the initial solution */
$T = T_{max}$; /* Starting temperature */
Repeat
  Repeat /* At a fixed temperature */
    Generate a random neighbor s' ;
    $\Delta E = f(s') - f(s)$;
    If $\Delta E \leq 0$ **Then** $s = s'$ /* Accept the neighbor solution */
    Else Accept s' with a probability $e^{\frac{-\Delta E}{T}}$;
  Until Equilibrium condition
  /* e.g. a given number of iterations executed at each temperature T */
  $T = g(T)$; /* Temperature update */
Until Stopping criteria satisfied /* e.g. $T < T_{min}$ */
Output: Best solution found.

Fig. 1. Pseudo Código SA

V. Propuesta de Solución

La solución propuesta es un producto de datos en la forma de un tablero (*dashboard*) dirigido hacia al área de Planeación Lógistica.

El objetivo de este tablero es que el área cuente con una manera de visualizar las rutas óptimas. Para esto, se pretende que a partir de la selección que hace el usuario del estado y la fuerza de ventas, pueda visualizar la ruta en un mapa, los nodos que recorre y la distancia total. Asimismo, el tablero permitirá al usuario escoger entre diferentes algoritmos (*Particle Swarm Optimization* y *Simulated Annealing*) o los dos.

Para lograr esta solución, se construyó un *pipeline* a través del cual fuera posible encontrar las rutas óptimas de todas las fuerzas de venta y con ambos algoritmos. En detalle, dicho pipeline toma los datos de los nodos de una fuerza de venta que están almacenados en una base de datos, ejecuta el algoritmo en paralelo (*Particle Swarm*) o secuencial (*Simulated Annealing*) y guarda los resultados en la base de datos.

Implementación en paralelo. La paralelización del algoritmo se llevó a cabo utilizando Dask y en particular el decorador @dask.delayed; el uso de éste último permitió obtener funciones lazy, lo cual dio lugar a poder apilar o juntar estas funciones, de tal manera que fuera posible controlar el reparto de trabajo en paralelo. Dado que la paralelización del algoritmo ocurrió en los datos, al utilizar Dask se distribuyó la información de las fuerzas de venta de manera que cada trabajador (*core*) ejecutara el algoritmo de *Particle Swarm* un determinado número de fuerzas de ventas diferentes, cubriendo de esta manera el total de datos. La paralelización está implementada de forma que la distribución de las tareas (ejecución del algoritmo) se adapte al número de cores presentes en el equipo con el que se esté trabajando.

Una vez que los resultados se cargan en la base de datos, son consumidos por el tablero para su visualización.

Uso de Docker. Se construyeron dos imágenes de Docker diferentes y ambas están disponibles en DockerHub. La primera (`paolamedo/numerical_methods`) contiene las librerías para correr el *pipeline*; incluye bibliotecas para conectarse a la base de datos, correr en paralelo y correr los algoritmos. La segunda (`paolamedo/dash`), se utilizó para la creación del tablero.

Uso de AWS. Se construyó una base de datos (*Relational Database Service, RDS*) en la plataforma de Amazon Web Services (AWS). La base de datos tiene dos esquemas: *raw* y trabajo; las tablas contenidas en cada uno y su descripción se detallan a continuación:

Esquema	Tabla	Descripción
raw	fuerza_ventas	Datos originales con los nodos de cada fuerza de venta
trabajo	nodos	Catálogo con las coordenadas geográficas de los clientes y sucursales.
trabajo	resultados	Rutas óptimas y distancia mínima para cada fuerza de venta y algoritmo.

Tabla 2. Descripción de la base de datos

VI. Evaluación de los Algoritmos

Para evaluar el desempeño de los algoritmos se eligieron dos rutas, una con 6 nodos y la segunda con 10 nodos. Con el fin de poder comparar los resultados de ambos algoritmos, se decidió buscar los mejores hiperparámetros de cada algoritmo para posteriormente fijar dichos valores; y ejecutar cada algoritmo 100 veces. El detalle de este conjunto de pruebas puede encontrarse en evaluación de los algoritmos.

La evaluación de los mejores hiperparámetros requirió de la construcción de un método llamado *GridSearch*, el cual recibe a su vez los siguientes parámetros: el grafo de una fuerza de ventas fijo, un diccionario de hiperparámetros (los cuales varían dependiendo del algoritmo a utilizar), el algoritmo a evaluar y el número de iteraciones que se correrá por cada combinación de hiperparámetros. Este método evalúa el algoritmo con todas las combinaciones que se pueden generar a partir del diccionario de parámetros y, devuelve una tabla con la distancia mínima obtenida, la distancia máxima y la frecuencia relativa de la distancia mínima. Es importante enfatizar que la elección de los mejores hiperparámetros considera tanto la distancia mínima obtenida como la frecuencia relativa de la misma.

A. Evaluación con 6 nodos.

Dentro del conjunto de datos que se tienen disponibles, existen varias fuerzas de venta que deben recorrer 6 nodos. Se decidió elegir la fuerza de venta **80993** perteneciente al estado de Nuevo León para llevar a cabo estas pruebas.

A.1. Simulated Annealing.

La función que se utilizó para este algoritmo es `Annealer` del paquete `simanneal` de Python. Los hiperparámetros de entrada son:

- Tmax:Temperatura máxima con la que inicia el algoritmo
- Tmin: Tempertatura mínima a la que llega al equilibrio
- steps: Número de iteraciones
- updates: Número de actualizaciones

Se probaron 36 combinaciones diferentes las cuales se corrieron 100 veces para poder ver la variación entre iteraciones y la estabilidad del algoritmo.

Sobre el resto de los parámetros, se escogieron los siguientes valores alrededor e incluyendo los parámetros por omisión del algoritmo:

- Tmax:10,000 y 25,000
- Tmin: 1 , 2,5 , 5
- steps: 500, 5,000
- updates: 10, 50, 100

La distancia mínima y máxima obtenida: 5.604 km es la misma para las 36 combinaciones. Estos resultados pueden ser consecuencia de haber utilizando los parámetros por default o cercanos que ya fueron previamente optimizados, por eso se hizo el siguiente experimento: correr el algoritmo 100 veces solo con 1 paso, $updates = 2$, $Tmin = 1$ y $Tmax = 2$. En este caso, la distancia mínima de 5.604 sólo se alcanzó una vez y la distancia máxima fue de 8.532.

Sin embargo, la tabla 3 de frecuencia de rutas, refuerza que el algoritmo es estable con los parámetros óptimos. Para la fuerza de ventas con 6 nodos, las 100 simulaciones arrojan la misma ruta.

Ruta	Frec	km
1020235635-1020053072-1006681965-1001402004-...	100	5.604

Tabla 3. Rutas de SA con 6 nodos partiendo del nodo base 11037

A.2. Particle Swarm. Con respecto al algoritmo de *Particle Swarm*, se eligieron los siguientes hiperparámetros para la prueba de *GridSearch*:

- Iteraciones:10, 50, 100
- Partículas: 1, 5, 100
- α: 0.5, 1
- β: 0.5, 1

Lo anterior dio lugar a 36 combinaciones de hiperparámetros y cada combinación fue ejecutada 100 veces, dando un total de 3,600 corridas. En términos generales se identificó que la distancia mínima obtenida se mantuvo en todos los casos en 5.604 km, sin embargo la distancia máxima presentó variaciones entre 5.605 km y 8.559 km; este resultado evidencia que a pesar de utilizar los mismos hiperparámetros, el algoritmo puede dar resultados distintos en cada ejecución. Considerar un número de partículas bajo (1 ó 5) dio lugar a frecuencias relativas de la distancia mínima menores al 5 %. Puntualmente, al considerar 100 iteraciones y 100 partículas; y $\alpha = 1$ y $\beta = 0{,}5$ se observó que la frecuencia relativa de la distancia mínima fue máxima; por lo tanto los anteriores fueron elegidos como mejores hiperparámetros.

Con base en los resultados anteriores, se procedió a fijar los hiperparámetros en los mejores valores, es decir, en aquellos que maximizaron la frecuencia relativa de la distancia mínima; y se ejecutó el algoritmo 100 veces. Esta acción permitió realizar un análisis sobre las rutas calculadas por el algoritmo, cuyo detalle se presenta en la tabla 4. Particularmente, se obtuvieron 3 rutas distintas; 2 de ellas con la misma distancia mínima de 5.604 km y la tercera con una distancia de 5.659 km. La variación entres estas dos distancias es de 55 m. Es importante recalcar que en un 99 % de las ejecuciones se obtuvo la menor distancia.

Los resultados aquí obtenidos, demuestran que la salida del algoritmo de *Particle Swarm* puede variar en cada corrida, aún cuando se mantengan fijos los hiperparámetros. Correr el algoritmo una sola vez no garantiza que se obtenga la distancia mínima. Por otro lado, al ejecutar el algoritmo varias veces, es posible obtener distintas rutas cuya distancia sea la misma.

Ruta	Frec	km
1020235635-1020053072-1006681965-1001402004-1020402992	57	5.604
1020402992-1001402004-1006681965-1020053072-1020235635	42	5.604
1006681965-1020053072-1020235635-1020402992-1001402004	1	5.659

Tabla 4. Rutas de PS con 6 nodos partiendo del nodo base 11037

A.3. Comparación entre los métodos. Para esta fuerza de ventas con seis nodos, el algoritmo de *Simulated Annealing* (SA) es más estable que *Particle Swarm* (PS) ya que corriendo ambos 100 veces con los parámetros óptimos, PS arrojó varías rutas mientras que SA la misma. Asimismo, la ruta que SA construyó de manera estable las 100 iteraciones es también la ruta con la mínima distancia que encontró PS. Las 2 rutas de mínima distancia encontradas por PS sugieren la presencia de varios mínimos y es interesante notar que SA no encontró el otro mínimo. Es importante remarcar que los tiempos de ejecución de PS fueron mucho menores que los de SA.

B. Evaluación con 10 nodos.

B.1. Simulated Annealing. Se utilizaron los mismos parámetros que en la evaluación de 6 nodos, probando 36 combinaciones diferentes a lo largo de 10 iteraciones. La distancia mínima y máxima obtenida fue de 8.458 km para las 36 combinaciones, por lo que nuevamente no se observó ninguna combinación de parámetros que variara la distancia mínima o máxima del algoritmo.

Estos resultados hacen pensar que internamente el algoritmo continua utilizando los parámetros por default, por lo que se realizó el siguiente experimento: Correr el algoritmo 10 veces con los siguientes hiperparámetros:solo 1 paso, 2 updates, Tmin de 1 y Tmax de 2. Nuevamente se observó una distancia minima y máxima de 8,458km. Se podría pensar que esta estabilidad se debe a la reducida cantidad de nodos (10)con la que se esta trabajando.

Asimismo, la tabla 3 de frecuencia de rutas, refuerza que el algoritmo es estable con 6 nodos porque 100 simulaciones arrojan la misma ruta.

B.2. Particle Swarm. Con respecto a la implementación del algoritmo de *Particle Swarm* para un grafo con 10 nodos, se decidió elegir a la fuerza de venta con el ID 96298 perteneciente a la Ciudad de México.

Para la prueba de *GridSearch* con 10 nodos se eligieron los mismos hiperparámetros que para el caso de 6 nodos.

En esta prueba, la distancia mínima obtenida varía entre 8.458 km y 9.469 km, mientras que la distancia máxima varía entre 8.674 km y 11.876 km. Es importante considerar que al igual que en la prueba de 6 nodos, el algoritmo puede dar resultados distintos en cada ejecución.

Los siguientes hiperparámetros fueron con los que con mayor frecuencia se logró obtener la distancia mínima y por lo tanto fueron considerados como los mejores:

- Iteraciones: 100
- Particulas: 100
- α: 1
- β: 0.5

Con estos hiperparámetros se ejecutó el algoritmo de Particle Swarm 100 veces y se obtuvieron 17 rutas distintas, 2 de ellas con la misma distancia mínima de 8.458 km. En el 55 % de las ejecuciones se logró obtener la distancia mínima, obsérvese a continuación la siguiente tabla (solo se mostrarán las primeras 3 rutas).

Ruta	Frec	km
1020451581-1009791566-1020328100-1020088646...	29	8.458
1009790047-1020449326-1020249367-1020253076...	26	8.458
1020451581-1009791566-1020253076-1020328100...	14	8.559

Tabla 5. Rutas de PS con 10 nodos partiendo del nodo base 11078

Estos resultados demuestran que el algoritmo de *Particle Swarm* puede arrojar rutas distintas con el mismo costo (distancia mínima). Por otro lado, también es posible que arroje otras rutas cuya distancia no sea la mínima distancia obtenida durante todas las simulaciones.

Conclusiones:

- Al igual que en las pruebas con 6 nodos los resultados del algoritmo pueden variar en cada corrida, aún cuando se utilicen los mismos hiperpárametros.
- Aumentar el número de partículas nos da una mayor probabilidad de encontrar el mínimo.
- Los mejores hiperparámetros fueron los mismos que para la prueba de 5 nodos.
- Considerando las combinaciones de parámetros que ejecutamos para esta prueba, podemos decir que darle un mayor peso a α que a β para este problema nos garantiza mejores resultados.
- Para un mayor número de nodos, obtenemos un mayor número de rutas distintas.

B.3. Comparación entre métodos. Hola, Erick: Esto es el 5 % que nos falta :)

VII. Resultados de Implementación en Paralelo

Conforme a lo expuesto en la sección **V**. la implementación en paralelo de *Particle Swarm* depende del número de *cores* disponibles. La división de tareas es por tanto proporcional a dicho número y para un mejor entendimiento de lo que está sucediendo, la visualización de tal distribución se muestra en las figuras 2, 3 y 4. Resulta natural pensar que mientras mayor sea el número de *cores* disponibles, menor será el tiempo de ejecución del algoritmo. Estos resultados se pueden apreciar en la tabla 6.

Número de cores	Tiempo de ejecución
4	9min 13s
8	4min 18s
12	3min 1s

Tabla 6. Tiempos de ejecución en paralelo de PS conforme al número de cores

En efecto, el aumento del número de *cores* conlleva a una disminución en el tiempo de ejecución; que en el caso particular de aumentar de 4 a 8 cores, dicha disminución es del 53,35 %, mientras que al incrementar de 8 a 12 *cores*, se aprecia una disminución del 30 %. Estos resultados sugieren que la disminución en el tiempo de ejecución en paralelo no es linealmente proporcional al número de *cores*. El detalle de las ejecuciones puede revisarse en ejecución en paralelo.

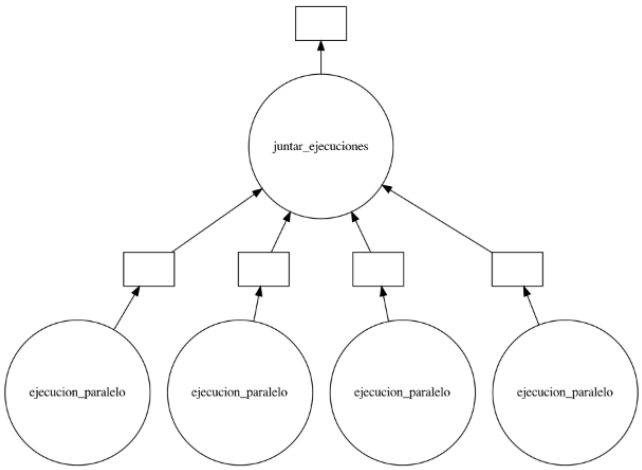

Fig. 2. División de tasks para PS utilizando 4 cores

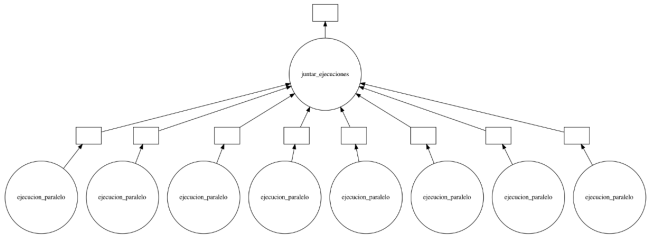

Fig. 3. División de tasks para PS utilizando 8 cores

Fig. 4. División de tasks para PS utilizando 12 cores

VIII. Visualización de rutas óptimas

Despliegue de aplicación. El tablero se construyó utilizando la herramienta "Dash". Dash es un entorno de trabajo de Python

que está basado en Flask y React. Se escogió este entorno de trabajo por la facilidad de implementar el código previamente escrito en Python para la creación de mapas y conexión a la base de datos.

El despliegue se pensó hacer a través de una EC2 pero se exploraron alternativas para reducir costos. Finalmente, se desplegó en Heroku ya que no implicaba un costo extra en el proyecto.

La figura 5 muestra la vista del tablero que está disponible en https://tcp-dashboard.herokuapp.com/. Del lado derecho, el usuario selecciona el algoritmo, la ciudad y la fuerza de ventas. Al darle click al botón "Ver ruta"se muestra el mapa con la ruta y en la parte inferior, la ruta con los nodos de los nombres y la distancia más corta. Asimismo, la figura 6 muestra la pestaña de más información con datos del proyectos y un vínculo al repositorio de GitHub.

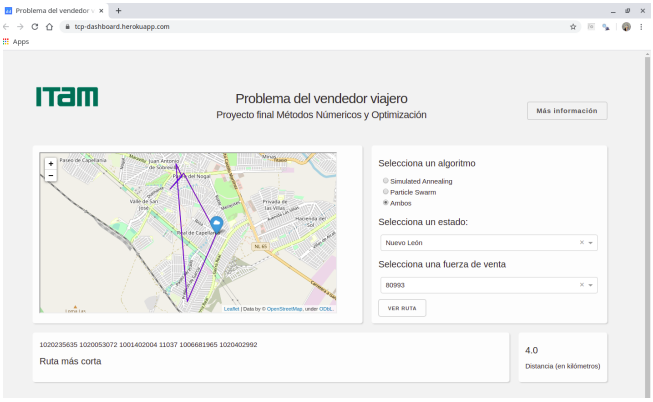

Fig. 5. Vista principal del tablero

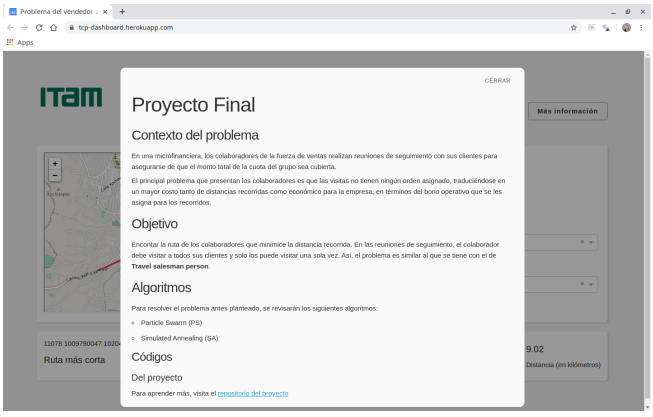

Fig. 6. Viñeta de más información

Ejemplos de visualización. La figura 7 muestra la ruta de la fuerza de venta **80993** del estado de Nuevo León para ambos algoritmos. *Simulated Annealing* es la línea roja y *Particle Swarm* la línea morada.

La figura 8 muestra la ruta de la fuerza de venta **96298** de la Ciudad de México para ambos algoritmos. De nuevo, *Simulated Annealing* es la línea roja, *Particle Swarm* la línea morada y la estación de partida está indicada por una nube.

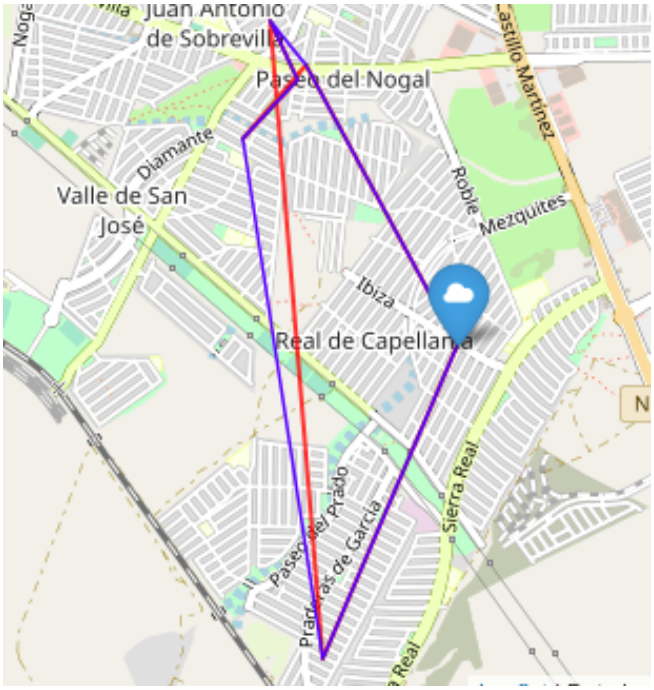

Fig. 7. Visualización de una ruta con 6 nodos

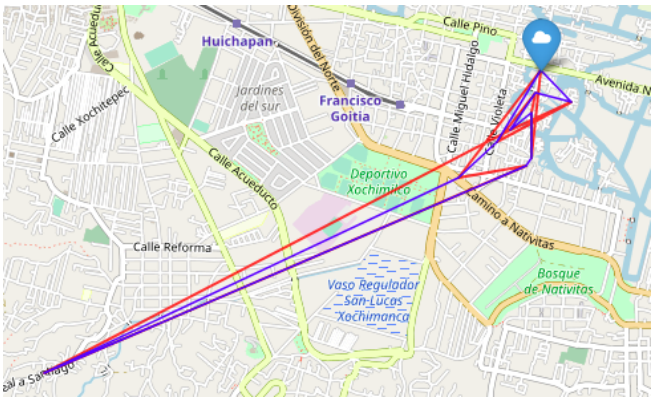

Fig. 8. Visualización de una ruta con 10 nodos

IX. Conclusiones

Como se menciona al inicio del documento, el presente trabajo tiene por objetivo encontrar la ruta con la menor distancia para los promotores de la fuerza de ventas sin hacer distinción del día de la semana en la que le toque visitar al cliente; es decir, se consideraron todos los puntos que un colaborador puede llegar a visitar. Para encontrar la mínima distancia, se estudiaron dos algoritmos que se han utilizado en el problema del TSP: *Particle Swarm Optimization* y *Simulated Annealing*. El primero se ejecutó en paralelo y la función fue desarrollada por el equipo; para el segundo algoritmo, se utilizó una función disponible ya en Python y, por la forma en la que está construida, no se pudo ejecutar en paralelo el algoritmo.

Referencias

[1] Castros, M. Solution to TSP (Travelling salesman problem) using Particle Swarm Optimization (PSO) - Language: Python disponible en https://github.com/marcoscastro/tsp_pso

[2] El-Ghazali, Talbi. Metaheuristics: From desing to implementation. John Wiley & Sons, Inc.2009.

[3] Günther Zäpfel, Roland Braune. Metaheuristic Search Concepts. Springer. 2010.

[4] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi.(1983). Optimization by Simulated Annealing. Science, New Series, 220(4598):671-680.

[5] Geo, P., Python module for Simulated Annealing optimization, disponible en https://github.com/perrygeo/simanneal

[6] De-Souza, X., et al. Coupled Simulated Annealing. 2007

[7] Sarman K., H., Arjun H., J., and Yogesh P., K. (2012). Solving City Routing Issue with Particle Swarm Optimization.International Journal of Computer Application, 47(15):27–30

[8] Xuesong, Y., Can, Z., Wenjing, L., Wei, L., Wei, C., and-Hanmin, L. (2012). Solve Traveling Salesman Problem UsingParticle Optimization Algorithm.International Journal of Computer Science, 9(2):264–271.