UNIVERSIDAD NACIONAL AUTONÓMA DE MÉXICO

PROGRAMA DE MAESTRÍA EN INGENIERÍA DE SISTEMAS-
INVESTIGACIÓN DE OPERACIONES

MODELO TSP PARA LA GENERACIÓN DE RUTAS DE VEHÍCULOS DE SEGURIDAD PÚBLICA

TESIS

PARA OPTAR POR EL GRADO DE

MAESTRA EN INGENIERÍA

PRESENTA

NILSE PAMELA ROMERO BASURTO

TUTORA PRINCIPAL:

DRA. ZAIDA ESTEFANÍA ALARCÓN BERNAL

FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA CD.MX., 2021

Junio

JURADO ASIGNADO:

Presidente: Dr. Ricardo Aceves García

Secretario: Dra. Patricia Esperanza Balderas Cañas

Vocal: Dra. Zaida Estefanía Alarcón Bernal

Primer suplente: Dra. Katya Rodríguez Vazquez

Segundo Suplente: Dra. Bibiana Obregón Quintana

Ciudad de México

Tutora de Tesis:

Dra. Zaida Estefanía Alarcón Bernal

Dedicatoria

A mi familia, los amo.
A mi padre Gustavo Romero Pérez por ser mi soporte incondicional y mi superhéroe. A mi madre Norma Mónica Basurto Higareda por ser esa persona llena de consejos, palabras de aliento, amor y además ser mi superheroína. A mi hermano Daniel Romero Basurto por estar siempre acompañándome y ser mi cómplice de locuras. Los amo
A mis abuelitos Esther y Blas por confiar y apoyarme siempre desde una palabra de aliento, un abrazo o un beso, los amo.
Y no menos importante a mi tía Ale por ser esa persona que se que siempre está detrás de mi apoyando, te quiero.

Agradecimientos

De manera especial y sincera a la Dra. Zaida Estefanía Alarcón B. por su tiempo, esfuerzo y dedicación en la creación de este trabajo y por todo lo que me han enseñado tanto académica como personalmente.
Al jurado por tomarse el tiempo de leer, revisar y corregir mi trabajo, especialmente al Dr. Ricardo Aceves García, a la Dra. Patricia E. Balderas Cañas, a la Dra. Katya Rodríguez Vázquez y a la Dra. Bibiana Obregón Quintana.
De igual manera quisiera agradecer a la Dra. Zaida E. Alarcón Bernal y al Dr. Ricardo Aceves García que desde hace 4 años me han ayudado a crecer de forma personal y acádemica, muchas gracias por todo lo que me han brindado.
Además, quisiera agradecer a la Coordinación General de Estudios de Posgrado (CGEP) por la beca otorgada para la realización y conclusión de mis estudios de maestría.
Y no menos importante a mis amigos que formaron parte de esta etapa y se volvieron lo mejor, quiero agradecer especialmente a Mónica Ocaranza y a Liliana Galván por ser mis amigas casi hermanas que me dejo esta etapa, gracias por siempre estar las quiero. A Osvaldo Palma por ser ese amigo incondicional que siempre me traía a tierra cuando mi trabajo no llegaba a un buen punto, muchas gracias. Y a César Guerrero quién después de un semestre se volvió el amigo que me tendió la mano y me ayudo a crear parte importante de la solución de este trabajo.

Índice general

Introducción

El aumento en los índices delictivos es un hecho que preocupa a la sociedad. De manera frecuente vemos como son cometidos delitos sobre bienes o pertenencias de los ciudadanos. Si bien existen medidas que buscan mitigar este fenómeno, no se ha encontrado alguna estrategia que pruebe tener mayor efectividad. Desde el punto de vista de un ciudadano, se observa que uno de los eslabones más débiles en el combate a la delicuencia es la falta de una estructura de actuación de las patrullas policiales, y al hacer una búsqueda en la literatura encontramos que no hay muchos estudios científicos que exploren esta área, por lo que, en este trabajo se propone una estrategia que podrían seguir los policías en la zona donde están asignados.

La estrategia se desarrolla con la propuesta de una modificación al problema del agente viajero para encontrar rutas policiales con base en los puntos de mayor incidencia delictiva y considerando un factor aleatorio para evitar la predictibilidad de la ruta. Este modelo se resuelve mediante un algoritmo metaheurístico bioinspirado y se prueba su funcionalidad con un caso de estudio en la Ciudad de México.

Justificación del proyecto

Actualmente existe una alta incidencia delictiva que afecta nuestra ciudad, una de las alternativas de solución ha sido aumentar el parque vehicular de las patrullas por delegación, sin embargo, muchas veces este incremento no muestra un impacto significativo en las incidencias por la falta de rutas efectivas para llegar de manera oportuna a los lu-

gares donde se presentan estos problemas. Esta situación nos motivó a presentar un modelo que permita establecer algunas rutas para la vigilancia en cada cuadrante de la ciudad.

Objetivos del proyecto

El objetivo principal de este trabajo es:

Desarrollar un modelo de rutas de patrullaje no predecibles que permita maximizar la cobertura en los puntos de mayor riesgo en cuadrantes de mayor incidencia delictiva.

Para lograr el objetivo descrito, se proponen los siguientes objetivos específicos:

- Identificar y ajustar un modelo adecuado para el sistema, considerando sus principales características, como zonas de alta incidencia delictiva, rutas de vehículos y la necesidad de evitar la predictibilidad.
- Identificar un método eficiente para el modelo propuesto.
- Desarrollar el método de solución con base en el modelo construido.
- Encontrar los datos actuales de incidencias delictivas y el número de patrullas por cuadrantes en Ciudad de México para poder desarrollar un caso de estudio que nos permita probar el modelo propuesto.
- Dentro del caso de estudio, proponer rutas aleatorias de patrullaje para un cuadrante de la Ciudad de México

A fin de lograr estos objetivos, este trabajo está estructurado de la siguiente manera: en el capítulo 1 se presentan algunos antecedentes del problema y una breve revisión del estado del arte. En el capítulo 2 se presenta el modelo del agente viajero que se utilizó para el desarrollo del modelo de rutas de patrullaje. En el capítulo 3 se describe el método

de solución propuesto para el modelo desarrollado. En el capítulo 4 se presenta el modelo del agente viajero modificado para la definición de rutas de patrullas y su aplicación en un caso de estudio del cuadrante policial Álamos 1 junto con los resultados obtenidos utilizando el modelo propuesto. Finalmente se presentan las conclusiones. En la última parte se presentan como anexos la matriz completa de distancias de los puntos del cuadrante de Álamos y la bibliografía utilizada.

Capítulo 1

Antecedentes

De acuerdo con datos recabados por el Observatorio Nacional Ciudadano[1] en el mes de diciembre de 2019 se sigue mostrando una tendencia a la alza en diversos delitos de alto impacto y pocos avances en el descenso de los delitos patrimoniales como robo a transeúnte, robo a casa habitación, robo de autopartes y automóviles, entre otros.

Se considera que el país aún carece de condiciones de seguridad efectivas para sus habitantes, tal y como lo demuestran, los incrementos en las tasas de diversos delitos tanto en la Ciudad de México como en el resto del país.

Una de las principales deficiencias es la falta de patrullajes por parte de la policía, así también como, el tiempo de respuesta ante una llamada de emergencia, por lo que este aspecto representa un área de oportunidad que puede abordarse desde el punto de vista de la optimización.

Uno de los problemas sociales más importantes a los que se enfrenta la sociedad mexicana actualmente es la inseguridad. De manera frecuente vemos como son cometidos delitos sobre bienes o pertenencias de los ciudadanos e incluso cómo cada vez son más sonados los crímenes que atentan contra la vida de las personas. De esta manera, intentando darle una solución, las secretarías encargadas de salvaguardar la integridad de los ciudadanos han buscado soluciones eficaces para dar frente a

[1] https://onc.org.mx/uploads/Mensual-dic19%20(1).pdf 09/1/2019 10:18 p.m.

este problema, una de estas soluciones ha sido el aumento del parque vehicular de patrullas, pero esto no se ha visto reflejado en las estadísticas de incidencia delictiva por diversas razones: la falta de denuncias de la ciudadanía ante la impunidad que se ha observado, situaciones sociales que hacen que todos estos eventos delictivos vayan en aumento, falta de respuesta rápida por parte de la policía [5], [16], entre otras razones, pero es en esta última donde nosotros nos vamos a centrar.

De la investigación realizada, no se tiene conocimiento sobre la existencia de alguna forma efectiva en la que las patrullas son distribuidas por zonas o cuadrantes en las diferentes ciudades o poblaciones mexicanas, asimismo, tampoco se tiene registro de alguna propuesta desde el punto de vista de optimización. La mayoría de trabajos trabajan con encuestas de satisfacción a la población [16], desarrollo de políticas públicas de seguridad para el desarrollo de indicadores [35], creación de manuales para la capacitación de las autoridades en la vigilancia del espacio urbano [11] o trabajo enteramente teóricos que evaluan la aplicación de la tecnolgía en la prevención del delito [1], por lo que en este trabajo se propone una asignación de rutas que ayude a llegar de manera eficaz a los puntos que tienen un mayor registro de incidencia delictiva. Es importante notar que debemos de tener en consideración que estas rutas no deben ser predictivas para los delincuentes.

Por lo tanto, buscamos analizar la asignación de rutas de las patrullas en los distintos cuadrantes o sitios de la ciudad, considerando a los puntos de mayor incidencia delictiva como los puntos de recorrido de las patrullas, de tal manera que se minimice la distancia total de las patrullas o de los policías y se pueda tener una mayor cobertura que garantice una mayor eficiencia, además de añadirle un valor aleatorio que me ayude a que las rutas sean predecibles.

Una forma de abordar este problema es utilizando el Problema del Agente Viajero (TSP), ya que, este problema nos da la posibilidad de visitar todos los puntos de incidencia delictiva en la menor distancia o en el menor tiempo posible considerando que estos puntos se visitarán una vez para cada ruta propuesta.

Estado del Arte

En el artículo [31] se hace una revisión de los problemas de ruteo y sus variaciones que se habían presentado hasta el 2011. En el se destaca que el primer problema planteado del tipo de ruteo de vehículos es el problema del agente viajero o TSP por sus siglas en inglés el cuál fue introducido por Flood en 1956, a partir de esta propuesta nacen muchas variaciones entre ellas el TSP generalizado en 1959 por Dantzing y Ramser, el cual modela el despacho de combustible a través de una flota de camiones a diferentes estaciones de servicio. También destaca el desarrollo del TSP múltiple en 1960 por Miller, Tucker y Zemlin el cual es una generalización en la cuál se tiene un depósito y m vehículos, es decir m agentes viajeros.

En [25] se encuentra una revisión a las investigaciones más recientes, hasta 2020, de las aplicaciones del problema del agente viajero (traveling salesman problema TSP) y el uso de los métodos metaheurísticos bio-inspirados para su solución. Donde se puede observar la investigación de algunas variaciones de este problema como es el *problema asimétrico del problema del agente viajero* el cual nos indica que los costos asociados al arco de un nodo son diferentes dependiendo la dirección que se tome $c(i,j) \neq c(j,i)$ donde i y j son nodos que se deben visitar. *El problema múltiple del agente viajero* toma en cuenta que existe un conjunto m de agentes disponibles que pueden visitar cierta cantidad de ciudades. *El problema del agente viajero con ventanas de tiempo* donde su idea básica es que el costo de viajar entre un nodo y otro depende del tiempo o *el problema del agente viajero generalizado* donde los nodos son particionados en diferentes conjuntos llamados clusters. Además, en este artículo se hace una revisión a los trabajos realizados con distintos algoritmos bio inspirados como el algoritmo genético, colonia de hormigas, cúmulo de partículas, algoritmo de la libélula, así como nuevos modelos bio-inspirados propuestos.

En [9] se presentan cuales son las aplicaciones, así como, el modelado del problema de los m-viajeros que es el tema que se aborda en el trabajo. Además, de que hace un revisión a los textos en donde se habla de nuevas propuestas para la solución del MTSP.

Respecto a la planificación de rutas en una ciudad pequeña, en [40] se

presenta un nuevo algritomo heurístico el cual ayuda a desarrollar rutas para poder llegar de un punto a otro dentro de una ciudad tomando rutas cortas, rutas más seguras y una de acuerdo al deseo del usuario las cuales se planean en distintos horarios del día notando que el riesgo de ser víctima de un delito se incrementa en horarios por la tarde/noche.

En cuanto al problema concreto del uso del problema del agente viajero o del problema de ruteo para el problema de las patrullas de policías, [26] trabaja con una idea dentro del área de seguridad que es encontrar las rutas más seguras para ciudadanos de a pie en ciudades pequeñas. Dentro de este mismo rubro, se encontró el artículo [15] donde presenta un diseño para los cuadrantes en una ciudad. Por lo que dentro del ámbito de ruteo enfocado en seguridad podemos concluir que no hay una gran variedad de trabajos publicados.

Asimismo, en [25] y [27] se tiene la revisión de algunos trabajos donde el Algoritmo Genético (AG) fue usado de distintas maneras para poder dar solución al problema del agente viajero en los cuales trabajan con los distintos métodos de selección que se utilizan dentro del AG, así como, la propuesta de modificaciones en los métodos de selección o la inicialización para la mejora del algoritmo. Además, algunos de ellos realizan una comparación de resultados con distintos métodos para ver cual de ellos puede tener un mejor desempeño.

En el aspecto de seguridad dentro del área de optimización, no se encontraron muchos trabajos, solamente en el área social se identificaron trabajos o manuales para el desarrollo efectivo de los policías, basado en encuestas de satisfacción por parte de los ciudadanos y de la experiencia de los policías. [16], [1], [11], [35]

Este análisis nos permitió desarrollar una propuesta para el despliegue de patrullas dentro de un área de una ciudad con alta incidencia delictiva tomando como base el modelo del agente viajero, y dada su complejidad, utilizar un método metaheurístico de solución. Ambos aspectos se desarrollan en los capítulos siguientes.

Capítulo 2

Metodología [2]

La estrategia metodológica que seguimos para la construcción del modelo de la modificación del problema del agente viajero fue el modelo del diamante. El modelo del diamante representa un proceso de solución de problemas, que a pesar de su simplicidad será de utilidad para desarrollar la estrategia que utilizaremos para abordar de forma lógica y coherente, los diferentes problemas que se presentan en la propuesta de las rutas policiales. De acuerdo con [34], el Modelo del Diamante se entiende como un sistema con un conjunto de elementos interconexos que forman el todo, y se caracteriza por tener las siguientes propiedades:

- Los atributos o actuación de cada elemento afectan los atributos y la conducta del todo.
- La afectación por parte de los elementos al todo depende de los atributos o actuación de al menos un elemento del conjunto.
- Todos los subconjuntos de elementos existentes tienen las características anteriores.

Esto nos indica que el todo es más que la suma de las partes, por lo que su separación es sólo con propósitos conceptuales desde el punto de vista estructural

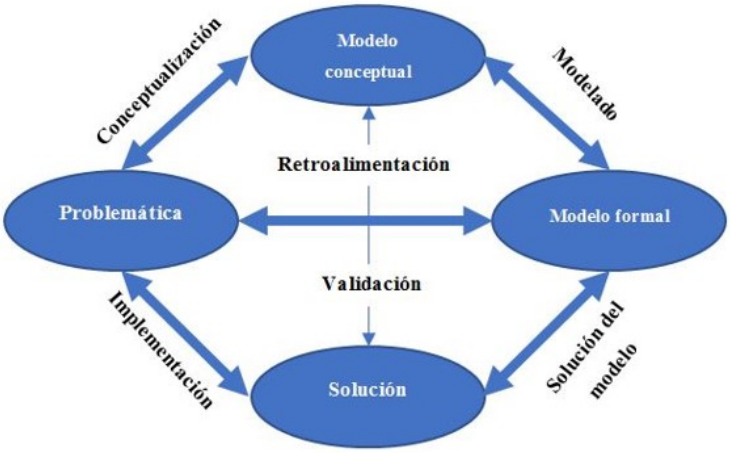


Figura 2.1: Modelo del diamante
Fuente: [2]

El proceso del modelo se establece a través de los siguientes subsistemas:

Situación Problemática. Toda forma de resolver problemas inicia con la existencia de una situación problemática o realidad, a través de la cual se plantea un reconocimiento de que las cosas están mal. Para la situación que deseamos analizar en este trabajo de las rutas policiales y el problema de la inseguridad se presenta en el capítulo de Antecedentes.

Modelo Conceptual. Es la representación que se utiliza como marco de soporte para ubicar y ordenar las percepciones. Para determinar alguna estructura del problema, definir el área de interés y decidir qué aspectos son relevantes y cuáles no. En nuestra propuesta, esto se hace en los Antecedentes donde se toma la decisión de utilizar el problema del agente viajero, ya que, dentro de las herramientas disponibles dentro de optimización este problema nos permite visitar cada uno de los puntos propuestos de incidencia delicitva de tal manera que podamos asegurar que existe una mayor cobertura, además de que es un problema facilmente manipulable que nos permite hacer modificaciones para generar aleatoriedad sin modificar el objetivo principal de encontrar una ruta que pase por todos sus puntos, y en los capítulos del Proble-

ma del Agente Viajero y del Algoritmo Genético, donde se explica a mayor profundidad el problema que decidimos utilizar y el algortimo de solución elegido.

Modelo Formal. Establecido el modelo conceptual, se procede mediante la abstracción, elaborar uno o varios modelos formales. El modelo formal generalmente consiste en un conjunto de reglas y/o símbolos elaborados bajo cierto sistema teórico, que requieren de habilidades analíticas y de abstracción, para establecer las relaciones y variables significativas. Esto se planteará en el capítulo de Modelo Propuesto Para el Diseño de Rutas de Patrullas donde desarrollamos el modelo, así como, proponer un ejemplo de aplicación real en la alcaldía Benito Juárez.

Solución. En esta actividad se pretende deducir las consecuencias de elegir diferentes cursos de acción, para apoyar la toma de decisiones e integrar las estrategias de cambio. Con la solución se pretende dar una única explicación del fenómeno, en congruencia con las ideas que el modelo conceptual tiene del problema y soluciones. Aplicando las herramientas propuestas, se obtienen resultados.

Si bien este modelo sistémico representa el proceso de resolución de problemas, que se entiende como un conjunto de elementos interconexos que forman el todo, es posible iniciar el proceso de solución de problemas desde cualquier subsistema, ya que cada uno de éstos enfatiza una parte del proceso que se sigue para resolver un problema y muestra que existen diversas formas de investigar. Por lo cual es posible utilizar alguno de los subsistemas como marco de referencia.

Capítulo 3

Problema del agente viajero

El problema del agente viajero es uno de los problemas de optimización combinatoria mas famosos y más estudiados. Como su nombre lo dice, el problema trata sobre un vendedor que tiene que visitar n cantidad de ciudades, la única condición es que solo puede visitar cada ciudad una sola vez y regresar al punto de inicio. Es importante notar que este problema ha tenido múltiples aplicaciones que no tienen nada que ver con un agente de ventas, un ejemplo, es cuando un camión sale de un centro de distribución para entregar bienes a cierta cantidad de ubicaciones o la planeación de la visita de los museos de la Ciudad de México de tal manera que se minimice el tiempo y se pueda visitar una vez cada museo.

3.1. Antecedentes históricos

El problema del agente viajero o the travelling salesman problem (TSP) tiene su origen en 1832 en el manual donde se proponían 5 circuitos para visitar Alemania y en uno de ellos parte de Suiza (Figura 3.1). Pero fue hasta 1940 que se le dio una representación matemática a los resultados del problema, de acuerdo con la pregunta sobre el camino más corto en un conjunto de puntos en un espacio métrico. [37]

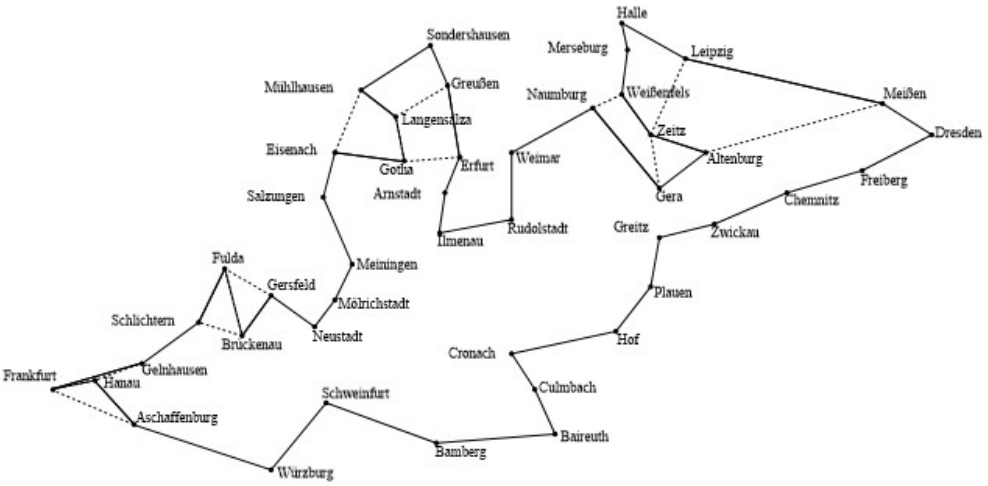


Figura 3.1: Un circuito de 45 ciudades Alemanas, como se describe en el manual del agente viajero de 1832.
Fuente: Schrijver, A. (1960). On the history of combinatorial optimization

En 1954 Dantzig, Fulkerson y Johnson llevaron a cabo uno de los progresos fundamentales para el desarrollo del problema del agente viajero en su artículo "Solution of a large scale traveling salesman problem" [8] problema en el cual propone una formulación matemática del problema del agente viajero con la cual pudieron encontrar una aproximación para recorrer 49 ciudades de Washington D.C.

Además, como se mencionó anteriormente se tienen diversas variaciones del problema del agente viajero, como el problema asimétrico, el problema múltiple del agente viajero y el problema del agente viajero con ventanas de tiempo o el problema del agente viajero generalizado. [25]

El problema del Agente Viajero se puede formular matemáticamente de distintas formas de acuerdo a cada una de las variaciones que tiene el problema. A continuación se presentan algunas.

3.2. Planteamiento del problema

Una versión del problema del agente viajero maneja la ruta como un ciclo, donde el agente no tiene permitido visitar una misma ciudad dos veces (excepto la ciudad donde se encuentra la oficina central donde empieza y termina su viaje), el ciclo contiene todos los vértices de la gráfica y se conoce como el ciclo de Hamilton en una gráfica. El origen del término es por el juego inventado en 1857 por Sir William Rowan Hamilton basado en la construcción de todos los ciclos contenidos en la gráfica de un dodecaedro. [4]

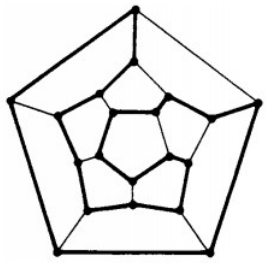


Figura 3.2: Circuito Hamiltoniano
Fuente: Béla Bollobás. (1998). Modern Graph Theory. USA: Springer.

Entonces, la teoría de gráficas es importante en el estudio del problema del agente viajero y podemos mencionar los aspectos importantes del problema. [13] Una gráfica $G = [V, E, D]$ donde:

- V un conjunto de vértices que representan las ciudades que se desean visitar.
- E un conjunto de aristas con pesos que modela las conexiones entre dichas ciudades.
- D una matriz que representa los valores de distancia del viaje entre cada par de ciudades adyacentes.

La segunda forma para representar el problema del agente viajero es mediante la programación matemática, que como se mencionó anteriormente fue propuesta por Dantzig, Fulkerson y Johnson [8].

$$min \quad \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{3.1}$$

$$s.a: \quad \sum_{j\in\Delta^+(i)} x_{ij} = 1 \qquad \forall i \in V \tag{3.2}$$

$$\sum_{i\in\Delta^-(j)} x_{ij} = 1 \qquad \forall j \in V \tag{3.3}$$

$$\sum_{i\in S, j\Delta^+(i)/S} x_{ij} \geq 1 \qquad \forall S \subset V \tag{3.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{3.5}$$

donde:

- x_{ij} representa una variable binaria que nos indica si el arco (i,j) está en la solución.
- c_{ij} es el valor de la distancia asociado al arco (i,j)

La ecuación 3.1 asociada a la función objetivo del problema nos representa la minimización de la suma de la distancia total de los arcos utilizados en el problema. Las ecuaciones 3.2 y 3.3 están asociadas a las restricciones del problema que nos indican que el agente debe entrar y salir de cada uno de los nodos exactamente una vez. Y la ecuación 3.4 se encuentra asociada a la restricción de que no deben existir ciclos que no cuenten con la cantidad completa de vértices dentro de la solución.

Con esta formulación, puede verse la naturaleza combinatoria del problema y la dificultad para encontrar la solución en la medida que aumenta el número de nodos. Si bien, esta es la formulación más famosa que existe del problema del agente viajero, la restricción 3.4 se puede expresar de muchas maneras diferentes entre ellas se encuentran las dos siguientes:

$$\sum_{i,j\in S} x_{ij} \geq |S| - 1 \qquad \forall S \subset V, 2 \geq |S| \geq n-2 \tag{3.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{3.7}$$

El problema puede formularse con una cantidad polinomial de restricciones, agregando variables reales u_i para $i = 1, ..., n$ y sustituyendo la restricción 3.4 por:

$$u_i - u_j + nx_{ij} \geq n - 1 \tag{3.8}$$

$$\forall (i, j) \in E, i \neq 0, j \neq 0 \tag{3.9}$$

Las cuales fueron propuestas por Miller, Tucker y Zemlin y fuerzan a que las variables reales u determinen una cantidad estrictamente creciente a lo largo de la ruta [24].

Este problema ha suscitado mucha atención porque es muy fácil de describir, sin embargo, es muy difícil de resolver. El problema puede resolverse analizando cada ruta de ida y vuelta en la gráfica para determinar la más corta. Sin embargo, a medida que aumenta el número de destinos, el correspondiente número de viajes de ida y vuelta crece rápidamente. Con 10 destinos, puede haber más de 300.000 permutaciones y combinaciones de ida y vuelta. Con 15 destinos, el número de rutas posibles puede superar los 87.000 millones.

3.3. Complejidad Computacional del TSP [29]

El problema del agente viajero se puede complicar rápidamente al aumentar la cantidad de ciudades o nodos, de tal manera que si tenemos un número de n ciudades con arco que los una, el número de rutas factibles que se debe considerar es de $(n-1)!/2$ esto porque existen $(n-1)$ posibilidades para la primera ciudad después de la ciudad de donde parte el agente, para la siguiente ciudad las posibilidades son de $(n-2)$ y así de manera sucesiva.

Es un problema que a lo largo del tiempo ha sufrido múltiples modificaciones de manera que se puedan obtener mejores soluciones, pero esto ha traído a discusión que tan fácil es resolverlo con método exactos, por esta razón el problema se conoce como $NP - Hard$ o $NP - Duro$, un tipo de problema computacional.

Se considera que un problema computacional tiene una buena solución cuando existe una algoritmo que pueda brindar una solución exacta de manera eficiente.

Un algoritmo se considera eficiente y útil para resolver un problema computacional siempre que su tiempo de ejecución crece como una función polinomial.

Asimismo, un problema combinatorio, como el problema del agente viajero, se puede dividir en 3 tareas computacionales diferentes:

- Optimización: Brinda representaciones para los parámetros relacionados con la cantidad de ciudades o vértices del problema y así ver la viabilidad.
- Evaluación: Determina el costo de la solución factible.
- Decisión de un problema a minimizar: Nos ofrece un valor entero B para el cuál se genera la pregunta si la evaluación encontrada es mayor a el, de tal manera que solo podamos tener respuestas de si o no.

Para el problema del agente viajero, estos tres problemas se ven representados de la siguiente manera:

- Optimización: Encontrar el ciclo Hamiltoniano mas corto para la gráfica G.
- Evaluación: Se obtiene la ditancia más corta del ciclo Hamiltoniano
- Decisión de un problema a minimizar: Se hace la pregunta ¿Existe algún ciclo Hamiltoniano que tenga un valor menor o igual al valor B propuesto?

A estos problemas computacionales los podemos clasificar en problemas de clase P y clase NP de acuerdo a su dificultad computacional.

Los problemas P son problemas que pueden resolverse de manera eficiente mediante algoritmos que cuenten con tiempo polinomial, problemas de decisión considerados fáciles, como la ruta más corta, árbol

de expansión o la programación lineal.

Los problemas NP son problemas donde si y solo si la decisión del problema en algunas de sus solicitudes con respuesta *sí* se puede resolver en tiempo polinomial mediante un algoritmo no determinista.

Un algoritmo polinomial es un algoritmo para el problema computacional en el que existe una función polinomial que resuelve cualquier instancia del problema en el tiempo límite. Algunos de los problemas considerados dentro de esta clase son el problema de la coloración en gráficas, programación del ciclo hamiltoniano, entre otros.

Dentro de la clase de problemas NP tenemos los siguientes problemas:

- $NP - Completos$ se considera que un problema es NP completo cuando puede transformarse en un tiempo polinomial. Una de sus propiedades mas importantes es que si existe un algoritmo de tiempo polinomial para uno de estos problemas, existen algoritmos de tiempo polinomial para los demás problemas NP. Para probar que un problema es $NP - Completo$ debe cumplir dos cosas, estar en NP y demostrar que todos los demás problemas en NP se pueden transformar en tiempo polinomial a él.

- $NP - Duros$ se considera que un problema P es $NP - Duro$ si todos los problemas en NP son polinomialmente transformables a P, pero su pertenencia a NP no se puede establecer. Aunque P es tan díficil como cualquier problema en NP no pertenece a los $NP - Completos$. También el problema $NP - Duro$ se utiliza para referirse a problemas de optimización (que no están en NP, porque no son problemas de decisión) cuyas versiones de decisión son problemas $NP - Completos$.

Como vimos, los problemas de alta dificultad computacional no cuentan con un algoritmo que nos brinde una solución óptima debido a que la cantidad de tiempo que se necesitaría para encontrar una solución es muy grande. Por lo que, para resolver estos problemas, se han propuesto métodos heurísticos y metaheurísticos.

3.4. Métodos de solución

3.4.1. Métodos exactos [19]

Un método exacto es aquel método de optimización que se utiliza para resolver un problema de optimización que garantiza el obtener un óptimo global en un tiempo finito y comprueba que efectivamente es el óptimo o si es el caso menciona si no existe una solución óptima para el problema.

Existen múltiples métodos exactos para resolver problemas combinatorios, como el problema del agente viajero, entre ellos se encuentra el algoritmo de ramificación y acotamiento incluida su variación de ramificación y corte, la programación dinámica, la búsqueda hacia atrás o backtracking y algunos más. Estos métodos se pueden ver como procesos de búsqueda en árboles, ya que se empieza con una solución inicial y va verificando cada uno de los subárboles del espacio de solución.

Uno de los métodos exactos más utilizado para la solución del problema del agente viajero, desde el punto de vista de su formulación de programación lineal, es el algoritmo de ramificación y acotamiento (Branch and Bound BB) en el cual se realiza una relajación en las limitaciones del problema y recupera la viabilidad mediante un proceso iterativo. Y la calidad de la solución que nos brinda el algoritmo este asociado con el límite que se tome en cuenta dentro de la relajación.

Utilizando la formulación presentada en la sección 2.2 en la restricción 3.4 se puede obtener un límite inferior inicial de manera que se tiene un problema de asignación que se puede resolver de forma exacta.

Ahora, hemos visto que el problema del agente viajero tiene un lugar central dentro de la investigación de operaciones, los problema que involucran pocos vértices se pueden resolver con los métodos que enunciamos, sin embargo, cuando se tienen más de 2000 vértices se pueden resolver utilizando la relajación pero su tiempo computacional es demasiado largo por lo que se han propuestos métodos, que si bien no nos dan la solución óptima, si nos dan una mejor solución al problema propuesto.

3.4.2. Métodos heurísticos

De acuerdo a Zanakins y Evans [41] un heurístico lo podemos definir como un "procedimiento simple, a menudo basado en el sentido común, que se supone ofrecerá una buena solución (aunque no necesariamente la óptima a problemas difíciles, de un modo fácil y rápido".

Los algoritmos heurísticos se pueden clasificar como [20]:

- Métodos constructivos: se caracterizan por construir una solución definiendo diferentes partes de ella en pasos sucesivos.
- Métodos de descomposición: dividen al problema en varios más pequeños y la solución la podemos obtener a partir de la solución de cada uno de estos.
- Métodos de reducción: tratan de identificar alguna característica de la solución que permita simplificar el tratamiento del problema.
- Métodos de manipulación del modelo: La solución la obtenemos haciendo alguna modificación de simplificación al problema original, ya sea, quitando restricciones, linealizando el problema, buscando alguna relajación, etc.
- Método de búsqueda de entornos: Obtenemos una solución inicial del problema original y se le van realizando modificaciones de forma recursiva para obtener una solución final, además de que se va a contar con un conjunto de soluciones vecinas que son candidatas a ser una nueva solución. Dentro de este conjunto es que vamos a encontrar a las técnicas metaheurísticas.

Algunos tipos de métodos heurísticos para resolver el problema del agente viajero son:

- Heurísticos del vecino mas cercano
- Heurísticos de inserción
- Heurísticos basados en árboles generadores

- GRASP

Otra clase de métodos de solución son los métodos metaheurísticos.

3.4.3. Métodos Metaheurísticos

Definimos a las metaheurísticas como procedimientos de búsqueda que no garantiza obtener el óptimo del problema solo nos brinda aproximaciones. Una de sus características, al contrario de los heurísticos, es que trata de huir de los óptimos locales. [32]

Para estos algoritmos se debe establecer un balance adecuado entre dos características contradictorias del proceso:

- Intensificación: cantidad de esfuerzo empleado en la búsqueda en la región actual, es decir, se busca que exista una explotación del espacio.

- Diversificación: cantidad de esfuerzo empleado en la búsqueda en regiones distantes del espacio, es decir, buscamos que se pueda explorar la mayor parte del área.

Este balance es necesario para poder identificar regiones que nos puedan brindar mejores soluciones y al mismo tiempo disminuir el tiempo de exploración.

Para las metaheurísticas no existe una clasificación establecida, pero hasta el momento se tienen tres grupos generales:

- Metaheurísticas constructivas: se parte de una solución vacía y se van añadiendo componentes hasta construir una solución

- Metaheurísticas basadas en trayectorias: parten de una solución inicial y de manera iterativa tratan de reemplazarla con alguna disponible dentro de la vecindad definida.

- Metaheurísticas basadas en poblaciones: Se parte de un conjunto de soluciones llamada población que van evolucionando de manera iterativa.

Algunos tipos de métodos metaheurísticos son:

- Búsqueda Tabú
- Recocido Simulado
- Algoritmos Genéticos
- Algoritmos Bioinspirados como la Colonia de hormigas (ACO), Cúmulos de partículas (PSO), entre otros.

Dadas las características del modelo que se presenta en este trabajo, se propone el uso de un método metaheurístico para encontrar una buena solución. Concretamente, se propone el uso de un algoritmo genético, que se describe a continuación.

Capítulo 4

Algoritmo Genético

La computación evolutiva imita de manera parcial los mecanismos de la evolución biológica basada en la teoría de Charles Darwin que a lo largo de estos miles de años los seres vivos se han visto sometidos a diversas pruebas para poder determinar cual es el mecanismo que permite que los individuos puedan adaptarse a su entorno lo que llevo a determinar que el material genético (ADN), el genotipo del individuo representa uno de los principales actores en este proceso. Pero además se requieren acciones que ayuden a este genotipo a evolucionar y adaptarse y cambiar sus características estructurales, fisiológicas o etológicas (fenotipo), estas acciones son la recombinación y mutación de los genes. Todos estos mecanismos actúan a lo largo de varias generaciones. De esta manera, la computación evolutiva imita de manera parcial estos mecanismos de evolución biológica. [33]

Las principales técnicas de la computación evolutiva son:

- Algoritmos genéticos
- Programación genética
- Estrategias evolutivas

Todos estos son conocidos como algoritmos evolutivos.

Cada uno de estos algoritmos evolutivos trabaja con individuos que pueden dar potenciales soluciones a un problema específico en estructuras de datos, esta posible solución corresponde a un genotipo a partir del cual se obtendrá un fenotipo con el que se evaluará a cada individuo. De esta manera lo que buscan estos algoritmos es generar, seleccionar, combinar y remplazar al conjunto de soluciones. En este trabajo nos centraremos en los algoritmos genéticos, estos fueron desarrollados por John Holland en 1975. [33]

Como se mencionó, los algoritmos evolutivos buscan imitar el proceso de la evolución biológica por lo que los algoritmos genéticos no son la excepción en este caso se centran en los cromosomas donde se encuentra la información del ser vivo que varía de una generación a otra. En la formación de un nuevo individuo se combina la información cromosómica de los progenitores; a este proceso lo llamaremos recombinación. Otra característica importante que se debe tomar en cuenta son las mutaciones que puede sufrir el cromosoma. [39]

Holland propuso mecanismos para la selección y cruza de estos individuos, esta propuesta se le conoce actualmente como el algoritmo genético simple o básico, en el cual se considera que los códigos genéticos están en binario. Los pasos que propuso son los siguientes [33]:

1. Codificación del dominio
2. Generar la población inicial de N individuos (Aleatoriamente).
3. Evaluar cada individuo de la población.
4. Selección a los individuos
5. Aquellos individuos seleccionados se combinan
6. Mutación de individuos
7. Obtener N individuos
8. Volver al paso 3, si no se cumple el criterio fin.

Si nosotros vemos estos pasos dentro de la evolución biológica nos referimos a lo siguiente [33]:

- Tenemos un conjunto de soluciones factibles que van a corresponder a miembros de una especie particular donde se va a medir la aptitud de cada miembro a partir del valor de la función objetivo.

- Ahora vamos a trabajar con el conjunto de soluciones factibles que denominaremos una población de prueba los cuales serían los miembros vivos de una especie, algunos de estos miembros más jóvenes y regularmente los más aptos sobreviven en la adultez y se convierten en padres y serán emparejados de manera aleatoria y después tienen hijos que van heredando características de ambos padres.

- Como los miembros más aptos de la población tienen una mayor probabilidad de convertirse en padre gracias a esto un algoritmo genético tiende a generar poblaciones mejoradas de soluciones de prueba.

- Otra cosa que puede ocurrir en este proceso natural son las mutaciones esto nos indica que los hijos pueden adquirir aptitudes que no posee ninguno de los padres. Dentro del algoritmo genético este fenómeno ayuda a que se pueda explorar una parte de la región factible quizá mejor de la que se había considerado anteriormente.

- Como último paso tenemos la supervivencia del más apto que tiende a conducir al algoritmo hacia una solución de prueba que es la mejor de todas las que se tenían disponibles y que se encuentra más cercana a una solución óptima.

En la figura 4.1 se presenta un esquema de los pasos del algoritmo genético

De acuerdo con estos pasos es importante tener los siguientes datos [3]:

- Representación cromosómica (genotipo)

- Población inicial

- Medida de evaluación

- Criterio de selección

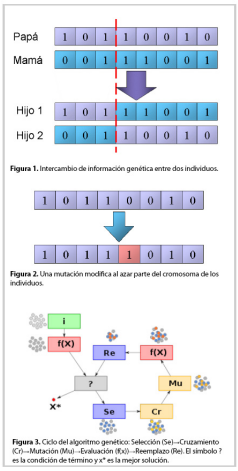


Figura 4.1: Algoritmo Genetico.
Fuente: CONOGASI, Ramón Garduño Juárez, Instituto de Ciencias Físicas, UNAM. Miembro de la Academia de Ciencia de Morelos

- Uno o varios operadores de recombinación
- Uno o varios operadores de mutación.

Además del algoritmo genético simple o básico, existen otros dos tipos específicos de este algoritmo:

- Algoritmo Genético Real
- Algoritmo Genético Entero

Es último es el que utilizaremos para este trabajo debido a que la solución del problema del agente viajero es expresada como un ciclo (permutación) donde los elementos que se desean visitar aparecen una sola vez, si bien el proceso que sigue es parecido al del simple, existen cambios en el proceso de cruza y mutación; para este proceso tomaremos en cuenta una permutación de un conjunto finito de elementos la cual es un arreglo de dichos elementos del conjunto. Dado n objetivos únicos, existen $n!$ permutaciones.

Para la **Cruza** contamos con dos vectores que llamaremos padres, de los cuales vamos a intercambiar cierta cantidad de elementos disponibles

para generar un nuevo vector de cruza.

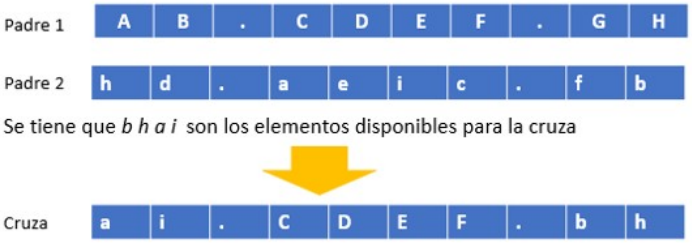


Figura 4.2: Cruza del Algoritmo Genético Entero.
Fuente: Presentación Algoritmo Genético, Semestre 2020-1, Rodríguez Vázaquez Katya [33]

Para la **Mutación** tomamos dos elementos del vector que llamamos padre y los intercambiamos de lugar.

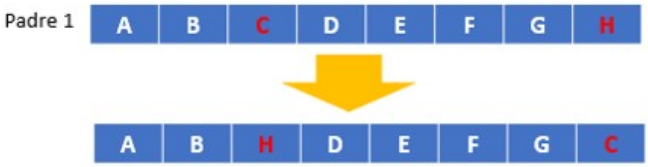


Figura 4.3: Mutación del Algoritmo Genético Entero.
Fuente: Presentación Algoritmo Genético, Semestre 2020-1, Rodríguez Vázaquez Katya [33]

Otro punto que nos interesa saber es que tan bueno o eficiente es este algoritmo metaheurístico para la solución del problema del agente viajero, por lo que para observar esto hacemos la comparación de este algoritmo contra algoritmos metaheurísticos de búsqueda y la programación lineal del problema, para poder ver esta comparación utilizamos distintos problemas ya existentes:

Ejemplo 1 [36]: P. Rose ha organizado un plan para guiar a 18 personas en un tour por todos los parques de beisbol de la liga Nacional. Para llevarlo a cabo esta planeando utilizar los ahorros de toda su vida por lo que quiere mantener los costos lo mas bajo posible. El recorrido iniciará y terminará en Cincinatti, para ello construyó la siguiente matriz con las

distancias, en millas, entre los estados donde se encuentran los parques de beisbol.

	Atl	Chi	Cin	Hou	LA	Mon	NY	Phi	Pit	Stl	SD	SF
Atl	0	702	454	842	2396	1196	864	772	714	554	2363	5679
Chi	702	0	324	1093	2136	764	845	764	459	294	2187	2187
Cin	454	324	0	1137	2180	798	664	572	284	338	2228	2463
Hou	842	1093	1137	0	1616	1857	1706	1614	1421	799	1521	2021
LA	2396	2136	2180	1616	0	2900	2844	2752	2464	1842	95	405
Mon	1196	764	798	1857	2900	0	396	424	514	1058	2948	2951
NY	864	845	664	1706	2844	396	0	92	386	1002	2892	3032
Phi	772	764	572	1614	2752	424	92	0	305	910	2800	2951
Pit	714	459	284	1421	2464	514	386	305	0	622	2512	2646
Stl	554	294	338	799	1842	1058	1002	910	622	0	1890	2125
SD	2363	2184	2228	1521	95	2948	2892	2800	2512	1890	0	500
SF	2679	2187	2463	2021	405	2951	3032	2951	2646	2125	500	0

Con estos datos realizamos la comparación de los siguientes métodos de solución:

- Programación lineal
- Búsqueda Local
- Recocido Simulado
- Algoritmo Genético

Las soluciones o mejores soluciones que encontramos se presentan a continuación, tomando en cuenta que, para los algoritmos metaheurísticos corrimos el algoritmo 10 veces, de ahí obtuvimos el resultado promedio y el mejor resultado que se presenta a continuación:

	Mejor resultado en una corrida	Resultados promedio en 10 corridas	Tiempo de ejecución
Búsqueda Local	7778	10530.9	0.3006
Recocido Simulado	7627	8463.4	0.3215
Algoritmo Genético	7577	7785.8	0.2813
Programación lineal[1]	7577		

Con estos resultados podemos observar que el algoritmo genético nos brinda una buena solución y en algunos casos es igual a la obtenida por el método exacto de programación lineal, además si lo observamos por el tiempo de ejecución de cada uno de los algoritmos nos damos cuenta que haciendo un total de 100 iteraciones en cada uno el algo-

ritmo genético muestra un tiempo de ejecución menor, de 0.2813 seg, mientras que en recocido simulado y en la búsqueda local es de 0.3215 y 0.3006 respectivamente.

Ejemplo 2 [28]: Problema de las 24 ciudades de Groetschel, este ejemplo forma parte de la librería de TSPLib de Gerhard Reinelt de la Universität Heidelberg [28]

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	257	187	91	150	80	130	134	243	185	214	70
2	257	0	196	228	112	196	167	154	209	86	223	191
3	187	196	0	158	96	88	59	63	286	124	49	121
4	91	228	158	0	120	77	101	105	159	156	185	27
5	150	112	96	120	0	63	56	34	190	40	123	83
6	80	196	88	77	63	0	25	29	216	124	115	47
7	130	167	59	101	56	25	0	22	229	95	86	64
8	134	154	63	105	34	29	22	0	225	82	90	68
9	243	209	286	159	190	216	229	225	0	207	313	173
10	185	86	124	156	40	124	95	82	207	0	151	119
11	214	223	49	185	123	115	86	90	313	151	0	148
12	70	191	121	27	83	47	64	68	173	119	148	0
13	272	180	315	188	193	245	258	228	29	159	342	209
14	219	83	172	149	79	139	134	112	126	62	199	153
15	293	50	232	264	148	232	203	190	248	122	259	227
16	54	219	92	82	119	31	43	58	238	147	84	53
17	211	74	81	182	105	150	121	108	310	37	160	145
18	290	139	98	261	144	176	164	136	389	116	147	224
19	268	53	138	239	123	207	178	165	367	86	187	202
20	261	43	200	232	98	200	171	131	166	90	227	195
21	175	128	76	146	32	76	47	30	222	56	103	109
22	250	99	89	221	105	189	160	147	349	76	138	184
23	192	228	235	108	119	165	178	154	71	136	262	110
24	121	142	99	84	35	29	42	36	220	70	126	55

	13	14	15	16	17	18	19	20	21	22	23	24
1	272	219	293	54	211	290	268	261	175	250	192	121
2	180	83	50	219	74	139	53	43	128	99	228	142
3	315	172	232	92	81	98	138	200	76	89	235	99
4	188	149	264	82	182	261	239	232	146	221	108	84
5	193	79	148	119	105	144	123	98	32	105	119	35
6	245	139	232	31	150	176	207	200	76	189	165	29
7	258	134	203	43	121	164	178	171	47	160	178	42
8	228	112	190	58	108	136	165	131	30	147	154	36
9	29	126	248	238	310	389	367	166	222	349	71	220
10	159	62	122	147	37	116	86	90	56	76	136	70
11	342	199	259	84	160	147	187	227	103	138	262	126
12	209	153	227	53	145	224	202	195	109	184	110	55
13	0	97	219	267	196	275	227	137	225	235	74	249
14	97	0	134	170	99	178	130	69	104	138	96	104
15	219	134	0	255	125	154	68	82	164	114	264	178
16	267	170	255	0	173	190	230	223	99	212	187	60
17	196	99	125	173	0	79	57	90	57	39	182	96
18	275	178	154	190	79	0	86	176	112	40	261	175
19	227	130	68	230	57	86	0	90	114	46	239	153
20	137	69	82	223	90	176	90	0	134	136	165	146
21	225	104	164	99	57	112	114	134	0	96	151	47
22	235	138	114	212	39	40	46	136	96	0	221	135
23	74	96	264	187	182	261	239	165	151	221	0	169
24	249	104	178	60	96	175	153	146	47	135	169	0

Al igual que con el ejemplo anterior las soluciones o mejores soluciones que encontramos se presentan a continuación, tomando en cuenta que, para los algoritmos metaheurísticos corrimos el algoritmo 10 veces con

un total de 100 iteraciones y en el caso del algoritmo genético 200 individuos con 100 generaciones, de ahí obtuvimos el resultado promedio y el mejor resultado que se presenta a continuación:

	Mejor resultado en una corrida	Resultados promedio en 10 corridas	Tiempo de ejecución
Búsqueda Local	1236	1324.6	0.30787
Recocido Simulado	1074	1176	0.28035
Algoritmo Genético	1340	1478.5	0.31162
Mejor solución encontrada	1272		

Con estos resultados podemos observar que para este caso el algoritmo genético no nos brinda una mejor solución, además de que incluso no tiene un mejor tiempo de ejecución, en este caso el algortitmo de recocido simulado tuvo un mejor desempeño.

Es importante destacar que los resultados de los métodos de búsqueda y del algoritmo genético se obtuvieron mediante la programación de los mismos utilizando Python con los editores de Spider y Jupyter Notebook, respectivamente. [2].

Estas pruebas se realizaron para probar el algoritmo genético que se programó y el cual se utiliza para la solución del modelo que se plantea en este trabajo, además de tener una idea de si nos podría dar buenos resultados y obtener un estimado del tiempo de ejecución que tendrá y la cantidad de individuos y generaciones que se utilizarán.

[2]Los códigos se presentan en los Anexos.

Capítulo 5

Modelo propuesto para el diseño de rutas de patrullas

Actualmente existe una alta incidencia delictiva que afecta nuestra ciudad, una de las alternativas de solución ha sido aumentar el parque vehicular de las patrullas, sin embargo, muchas veces este incremento no muestra un impacto significativo en las incidencias por la falta de rutas efectivas para llegar de manera efectiva a los lugares donde se presentan estos problemas. La estrategia se desarrolla con la propuesta de una modificación al problema del agente viajero para encontrar rutas policiales con base en los puntos de mayor incidencia delictiva y considerando un factor aleatorio para evitar la predictibilidad de la ruta.

El problema del agente viajero cuenta con muchas aplicaciones o variaciones distintas como lo es el problema del agente viajero asimétrico, el problema del agente viajero múltiple o el problema del agente viajero con ventanas de tiempo, entre otros. Pero dentro de estas variantes propuestas ninguna brinda la idea de una aleatoriedad o la no predictibilidad que buscamos para las rutas que recorran las patrullas en los cuadrantes policiales que le correspondan, por lo que para desarrollar el modelo de las rutas de patrullas en cuadrantes policiales se utilizará el problema del agente viajero para minimizar la distancia que van a recorrer las patrullas por todos los distintos puntos de incidencia delic-

tiva marcados en el cuadrante policial.

El objetivo del modelo propuesto es maximizar la cobertura en el cuadrante policial al proponer rutas aleatorias para que las patrullas destinadas a cada uno de los cuadrantes puedan hacer recorridos periódicos que no sean predecibles y así garantizar la seguridad de la ciudadanía. Para lograr plantear este modelo utilizamos el problema de programación lineal del agente viajero con una pequeña modificación para que se pueda cumplir la aleatoriedad que requerimos.

La modificación del modelo consiste en la generación de un vector aleatorio que multiplica al vector de costos en la función objetivo, De esta manera, la función objetivo cambia en cada momento que se resuelve. La propuesta consistiría en la solución del problema cada vez (por turno, o por día) para obtener un conjunto de rutas distintas para cada despliegue de patrullas.

El modelo planteado queda de la siguiente manera:

$$min \quad \sum_{(i,j)\in E} (r_{ij}c_{ij})x_{ij} \tag{5.1}$$

$$s.a: \quad \sum_{j\in\Delta^+(i)} x_{ij} = 1 \qquad \forall i \in V \tag{5.2}$$

$$\sum_{i\in\Delta^-(j)} x_{ij} = 1 \qquad \forall j \in V \tag{5.3}$$

$$\sum_{i\in S, j\Delta^+(i)/S} x_{ij} \geq 1 \qquad \forall S \subset V \tag{5.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E \tag{5.5}$$

donde:

- x_{ij} es una variable binaria que se asocia a cada una de las calles viables para ir del punto de incidencia delictiva i al punto j.
- c_{ij} es la distancia asociada a la calle que une los puntos i y j.
- r_{ij} es una matriz de valores aleatorios entre 0 y 1 asociado a cada uno de los arcos de la red.

- V el conjunto de puntos de incidencia delictiva.
- E el conjunto de arcos que unen a los puntos de la red.

La ecuación 5.1 asociada a la función objetivo del problema representa la minimización de la suma de la multiplicación del valor aleatorio por la distancia de los arcos utilizados en el problema, de tal manera que podamos garantizar que las rutas serán aleatorias. Las ecuaciones 5.2 y 5.3 están asociadas a las restricciones del problema que nos indican que la patrulla debe visitar cada una de los puntos de incidencia delictiva exactamente una vez y pasar al siguiente. Y la ecuación 5.4 se encuentra asociada a la restricción de que no deben existir ciclos que no cuenten con la cantidad completa de los puntos de incidencia delictiva dentro de la solución.

Para resolver este modelo propuesto se va a utilizar el Modelo Metaheurístico de computación evolutiva: Algoritmo Genético que se presentó en el capítulo anterior.

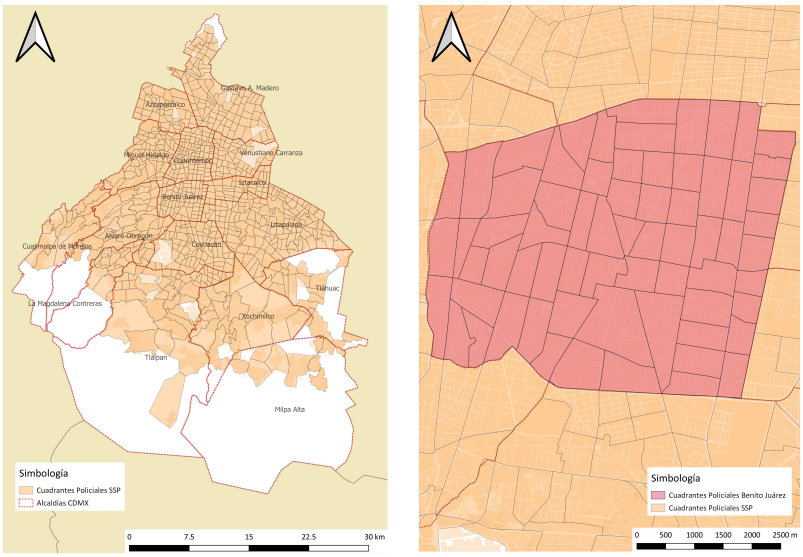
Para mostrar la aplicación del modelo y las soluciones que se obtienen, se utilizó, como ejemplo de aplicación, una demarcación de la Ciudad de México.

5.1. Aplicación

El aumento en los índices delictivos es un hecho que preocupa a la sociedad. De manera frecuente vemos como son cometidos delitos sobre bienes o pertenencias de los ciudadanos. Si bien existen medidas que buscan mitigar este fenómeno, no se ha encontrado alguna estrategia que pruebe tener mayor efectividad. Desde el punto de vista de un ciudadano, se observa que uno de los eslabones más débiles en el combate a la delicuencia es la falta de una estructura de actuación de las patrullas policiales, y al hacer una búsqueda en la literatura encontramos que no hay muchos estudios científicos que exploren esta área, por lo que, en este trabajo se propone una estrategia que podrían seguir los policías en la zona donde están asignados.

5.1.1. Cuadrantes Policiales de la Ciudad de México

En la Ciuidad de México existen 847 cuadrantes policiales definidos por la Secretaría de Seguridad y Proteción Ciudadana (SSPC) de la Ciudad de México (5.1), donde cada cuadrante cuenta con un punto de control y una cantidad específica de patrullas y de personal [1].



(a) Ciudad de México (b) Benito Juárez

Figura 5.1: Cuadrantes Policiales
Fuente: Elaboración propia con datos del INEGI [17]

A manera de ejemplo, en el trabajo vamos a centrarnos en uno de los 67 cuadrantes con los que cuenta la Alcaldía Benito Juárez (5.2): el cuadrante Álamos Sector 3.

Además, se hizo un análisis para saber cuales son los delitos con una mayor cantidad de carpetas de investigación en 2019 y 2020, esto debido a que manejar un total de 478,388 carpetas de investigación que incluyen 293 delitos diferentes. Dentro de este estudio encontramos que los delitos más cometidos son:

[1] Esta información se obtuvo del portal de datos abiertos de la Ciudad de México de la Fiscalía General de Justicia. [10]

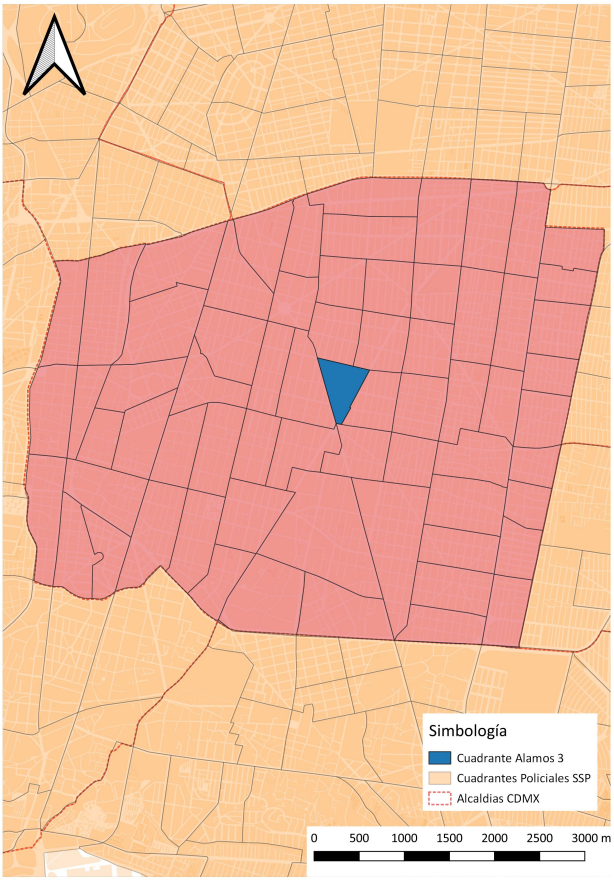


Figura 5.2: Cuadrante elegido de Benito Juárez
Fuente: Elaboración propia con datos del INEGI [17]

- Amenazas
- Robo a negocio con violencia
- Robo a transeúnte en vía pública con violencia.

De esta manera, dentro de la carpeta de investigación, tenemos los puntos donde fueron cometidos estos delitos dentro del cuadrante policial elegido, por lo que cada uno de estos puntos se convierte en un nodo de nuestra red los cuales serán visitados por la patrulla de ese cuadrante (5.3).

Cuadrante Álamos

Tenemos 20 nodos, 14 asociados a Amenazas, 1 con robo a negocio con violencia y 4 con robo a transeúnte en vía pública con violencia y 1 punto de control de la policía. Como se muestra en el mapa de la figura 5.3:

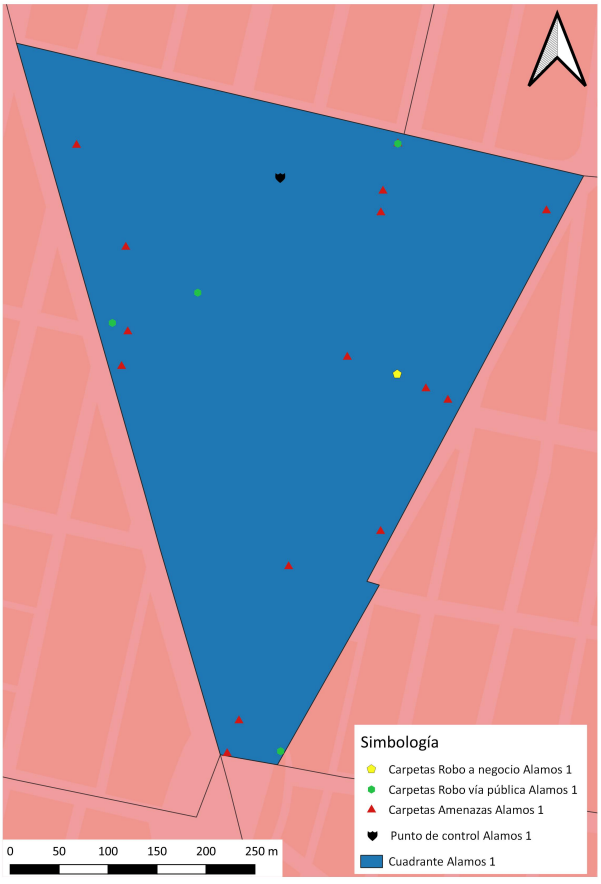


Figura 5.3: Cuadrante Álamos
Fuente: Elaboración propia con datos del INEGI [17]

Y tenemos un extracto de la matriz de distancias, estas distancias se obtuvieron tomando en cuenta las calles y el sentido de las mismas, que se utilizará para probar el modelo propuesto de rutas policiales [2] donde las abreviaturas utilizadas en los nodos son:

- PC: Punto de Control
- N: Robo a Negocio con Violencia
- VP: Robo a Transeúnte en Vía Pública con Violencia
- AM: Amenazas

	PC	N1	VP1	VP2	VP3	VP4	AM1
PC	0	1088.153	718.653	372.79	498.388	736.109	571.158
N1	326.223	0	1019.6	698.878	295.223	473.767	494.989
VP1	520.702	806.002	0	893.357	489.702	721.677	689.468
VP2	1130.711	1197.625	923.838	0	559.283	607.406	632.053
VP3	697.276	764.19	1160.29	954.446	0	580.119	199.611
VP4	736.468	537.972	946.048	1048.805	645.151	0	774.275
AM1	1050.616	1117.53	960.252	988.177	904.132	763.75	0

5.2. Resultados

Para poder obtener una propuesta de rutas policiales se utilizó el siguiente código del algoritmo genético [6]

[1]:
```
import numpy as np
import random
import operator
import pandas as pd
import matplotlib.pyplot as plt
from random import sample
```

[2] La matriz completa se encuentra en el Anexo D

La matriz de distancias de la aplicación (Se puede encontrar completa en el Apéndice E)

[2]:
```
                Dist = [[0, 1088.153, 718.653, 372.79,␣
↪498.388, 736.109, 571.158, 583.394, 718.653, 769.202,␣
↪791.187, 1005.731, 481.002, 711.536, 942.621, 1044.344,␣
↪1223.711, 1226.87, 776.479, 978.073],
                ...
```

Multiplicamos los valores aleatorios generados por la matriz de la distancias

[6]:
```
                D = np.matmul(test,Dist)
                print(D)
```

Declaramos la distancia total del problema y generamos la función de evaluación

[7]:
```
                def DistanciaTotal(Solucion):

                distancia = D

                DistanciaTotal = 0

                for i in range(0, len(Solucion)-1):

                DistanciaTotal +=␣
↪distancia[Solucion[i]][Solucion[i+1]]

                DistanciaTotal +=␣
↪distancia[Solucion[19]][Solucion[0]]

                return DistanciaTotal
```

Generamos la ruta inicial aleatoria

[8]:
```
def Ruta():
Cuidades = 20
ruta = []
for i in range(Cuidades):
ruta.append(i)
random.shuffle(ruta)
return ruta
```

Generamos la población inicial

[9]:
```
def PoblacionInicial(TamañoPoblacion):
poblacion=[]
for i in range(0,TamañoPoblacion):
poblacion.append(Ruta())
return poblacion
```

Se realiza la selección de la población

[10]:
```
def Seleccion(Poblacion,elit):
l=len(Poblacion)
selectionResults = []
df=pd.
↪DataFrame(list(range(l)),columns=['Individuo'])
df["Ruta"] = Poblacion
df["Aptitud"] = [DistanciaTotal(x) for x␣
↪in Poblacion]
df['Individuo']=list(range(l))#Creo el␣
↪dataframe
df=df.sort_values('Aptitud').
↪reset_index().drop('index', axis=1) #Reacomodar
df['cum_sum'] = df.Aptitud.cumsum()␣
↪#Selección de los mejores
```

```
                df['cum_perc'] = 100*df.cum_sum/df.
↪Aptitud.sum()

                for i in range(0, elit):
                selectionResults.append(df.iloc[i])
                for i in range(0, len(df) - elit):
                pick = 100*random.random()
                for i in range(0, len(df)):
                if pick <= df.iat[i,4]:
                selectionResults.append(df.iloc[i])
                break
                df=df.drop('cum_sum', axis=1).
↪drop('cum_perc', axis=1)[:]

                l1=list(range(0,l))  #Se realizan␣
↪permutaciones sobre los índices
                l2=list(range(0,l))
                random.shuffle(l1)
                random.shuffle(l2)
                I=[]   #Lista de índices seleccionados
                for i in range(0, len(Poblacion)):
                if df['Aptitud'][l1[i]] <␣
↪df['Aptitud'][l2[i]]:
                I.append(l1[i])
                else:
                I.append(l2[i])
                S1=df.iloc[I].sort_values('Aptitud').
↪reset_index().drop('index', axis=1)[:] #Reacomodar
                S1['Individuo']=list(range(l))

                return S1[0:elit] #Individuos ganadores␣
↪en formato dataframe
```

Función de cruza entre dos padres para crear al hijo

[16]:
```
                def Cruza(S,I):
                P=S['Ruta']
                F=[]
                l=len(P)
                for j in range(int(I/2)):  #Para generar␣
↪los padres con una probabilidad de cruce pc y un␣
↪aleatorio al
                pc = 0.6 #Probabilidad de cruce
                al = 0.8
                while al > pc:
                s=sample(range(l),2)
                al=np.random.uniform(0,1,1)[0]
                F.append(s)

                R=len(P[0])
                c1=int(R/3)       #Puntos de corte y de␣
↪casi del mismo tamaño
                c2=int((R-int(R/3))/2)+c1

                df=pd.DataFrame()
                Hijos=[]
                D=[]
                for i in range(int(I/2)):

                h1=P[F[i][0]][0:c1]+P[F[i][1]][c1:
↪c2]+P[F[i][0]][c2:R] #Estos son los hijos
                h2=P[F[i][1]][0:c1]+P[F[i][0]][c1:
↪c2]+P[F[i][1]][c2:R]

                e1=sorted(enumerate(h1),key=lambda x:x[1])
                e2=sorted(enumerate(h2),key=lambda x:x[1])
                E1=[e1[i][0] for i in range(len(e1)-1) if␣
↪e1[i][1]==e1[i+1][1]] #Indices de números repetidos␣
↪solo uno
                E2=[e2[i][0] for i in range(len(e2)-1) if␣
↪e2[i][1]==e2[i+1][1]] #Indices de números repetidos␣
↪solo uno
```

```
                EF1=[x for x in list(range(20)) if x not␣
↪in h1] #Elementos faltantes
                EF2=[x for x in list(range(20)) if x not␣
↪in h2] #Elementos faltantes
                H1=h1[:]
                H2=h2[:]

                for m in range(len(E1)):
                H1[E1[m]]=EF1[m]    #Cambio de elementos␣
↪repetidos por elementos faltantes para el hijo1

                for m in range(len(E2)):
                H2[E2[m]]=EF2[m]    #Cambio de elementos␣
↪repetidos por elementos faltantes para el hijo1

                Hijos.append(H1)
                Hijos.append(H2)
                D.append(DistanciaTotal(H1))
                D.append(DistanciaTotal(H2))

                df=pd.
↪DataFrame(list(range(I)),columns=['Individuo'])
                df["Ruta"] = Hijos
                df["Aptitud"] = D
                return df
```

[17]:

```
                def Mutacion(C):
                l=len(C)
                L=len(C['Ruta'][0])
                for i in range(l):
                pm=0.1
                pg=np.random.uniform(0,1,1)[0]
                if pg<pm:
                s=sample(range(L),2)
                C['Ruta'][i][C['Ruta'][i].
↪index(s[0])]=s[1]
                C['Ruta'][i][C['Ruta'][i].
↪index(s[1])]=s[0]
```

```
                return C
```

Se calcula la siguiente generación

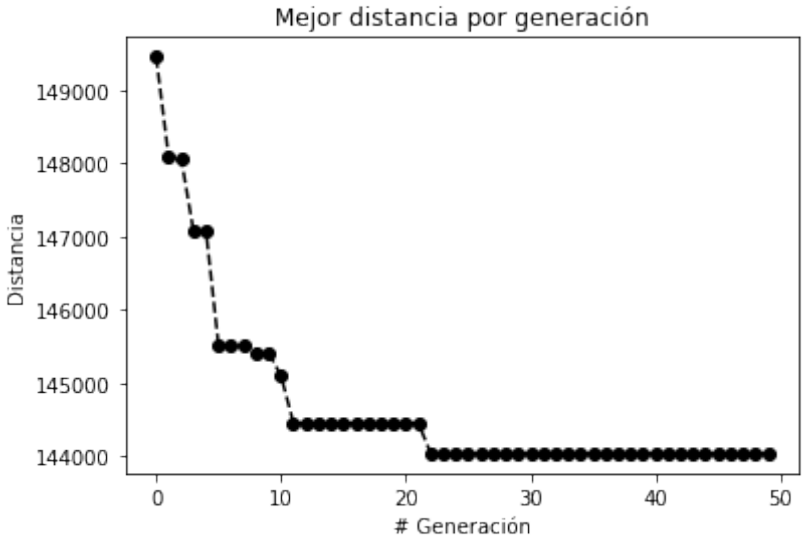
[18]:
```
                def SiguienteGeneracion(P, I, e):
                S=Seleccion(P,e)
                C=Cruza(S,I)
                M=Mutacion(C)
                return M
```

Algoritmo genético aplicando todas la funciones anteriores

[19]:
```
                def GATSP(Individuos, NGene, elit):␣
↪#NGene es el número de generaciones que se quieren

                SG=PoblacionInicial(Individuos)
                y=[]
                r=[]
                for i in range(NGene):
                SG = SiguienteGeneracion(SG, Individuos,␣
↪elit)
                M=SG.sort_values('Aptitud')
                y.append(M.iloc[0]['Aptitud'])
                r.append((M.iloc[0]['Ruta'],M.
↪iloc[0]['Aptitud']))
                SG=list(SG['Ruta'])
                plt.plot(np.arange(0, NGene, 1),y, '--ok')
                plt.title('Mejor distancia por␣
↪generación')
                plt.xlabel('# Generación')
                plt.ylabel('Distancia')
                plt.show()
                return r
```

[76]:
```
R = GATSP(Individuos=500,NGene=50,elit=10) #elit cantidad de individuos que se va a tomar de entre los mejores para seleccionarlos y cruzarlos
```



[77]:
```
len(R)
```

[77]:
```
50
```

[78]:
```
R[len(R)-1]
```

[78]:
```
([2, 19, 14, 16, 9, 5, 3, 15, 8, 4, 6, 12, 7, 18, 10, 13, 11, 17, 1, 0],
 144040.83132917102)
```

Al correr el código del algoritmo genético obtenemos distintas soluciones que nos dan las rutas y distancias que recorrerán las patrullas de tal manera que puedan visitar todos los puntos considerados focos

rojos cumpliendo el objetivo de proponer rutas aleatorias de patrullaje de manera que no sean predecibles para los delincuentes.

A continuación, se presenta una planeación de rutas para cada día de la semana con la distancia total que recorrerá la patrulla:

Tabla 5.1: Add caption

Día	Ruta	Distancia
Lunes	11-5-1-0-2-12-6-17-15-9-7-18-10-13-19-14-16-8-4-3	145,841.67
Martes	16-11-1-5-6-4-18-9-19-14-15-2-10-12-17-0-8-12-7-3	144,279.89
Miércoles	4-3-0-5-18-15-2-13-17-16-11-19-9-6-1-14-10-12-7-8	144,944.85
Jueves	4-7-6-15-10-12-5-17-0-2-18-11-14-19-1-9-13-16-3-8	144,674.75
Viernes	10-13-15-2-9-5-18-19-14-16-6-17-0-8-12-3-7-4-11-1	144,040.42
Sábado	17-0-2-5-6-10-12-7-18-19-14-11-1-4-9-13-16-3-15-8	144,732.47
Domingo	2-19-14-16-9-5-3-15-8-4-6-12-7-18-10-13-11-17-1-0	144,040.83

Ahora bien, como se detallo una de las acciones a tomar por parte del gobierno de la ciudad fue el ampliar el parque vehicular policial para poder ampliar la cobertura que tuvieran dentro de los cuadrantes y de esta manera se pudiera lograr tener una mayor percepción de seguridad, sin embargo, esto no basto para poder bajar los índices por lo que basados en esto quise plantear una alternativa que me permitiera modelar acciones con estas patrullas, lo cual me llevó a desarrollar el problema del agente viajero con valores aleatorios utilizando como vértices o nodos los puntos donde existe una carpeta de investigación por alguno de los delitos mencionados en la sección anterior y la conexión entre ellos la distancia real que existe entre cada uno de esos puntos y con esto generar rutas de patrullaje diario o incluso hacerlo por horarios asegurando que cada una de ellas es completamente diferente a la del día o urno anterior, por lo que en la tabla 5.1 se muestra el resultado de correr nuestro modelo mediante el algoritmo genético de solución que nos da las rutas de patrullaje para cada día de la semana, si bien es un modelo que nos da de forma eficaz una buena aproximación a rutas que se pueden utilizar dentro del cuadrante y es fácil de utilizar, la forma en que se obtuvieron los datos de la red para la aplicación es el punto que se podría optimizar o modificar de acuerdo a los datos que se tuvieran.

Capítulo 6

Conclusiones

Si bien siempre han existido estudios periódicos sobre la criminalidad, son pocos los que involucran modelos matemáticos.

Aún así, estos estudios pueden ser inexactos, ya que, no toman en cuenta el comportamiento humano o diversos factores que pueden alterar el estudio. En este trabajo se decidió usar el problema del agente viajero para buscar respuesta o posibles alternativas para el rápido accionar de las patrullas dentro de los cuadrantes policiales.

Para el desarrollo de este trabajo se analizaron un conjunto de modelos para identificar el más adecuado para hacer una propuesta a la problemática identificada. Después de un profundo análisis, se tomó como base el problema del agente viajero para encontrar distintas rutas de patrullaje en cuadrantes policiales, poniendo especial atención en los cuadrantes donde la incidencia delictiva es mayor.

Una vez seleccionado el modelo, se tuvo que adaptar a las circunstancias del problema: debían considerarse como nodos los puntos de mayor incidencia delictiva, y con eso se determinó la función de costo, sin embargo, en un caso ideal (sin vínculos de la policía con los delincuentes), no podía mantenerse la misma ruta todo el tiempo, ya que podría volverse una forma de comisión de delitos, al saber dónde se encontraría la patrulla en cada momento. Con esta consideración se añadió una matriz de valores aleatorios que permitiera la generación de rutas

distintas.

Al tratarse de un modelo de agente viajero, un problema NP-Duro, el uso de métodos de solución exactos, considerando el tamaño del ejemplo que se pretendía resolver, no era adecuado, por lo que se propuso el uso de un algoritmo genético implementado en Python, que se probó comparando las soluciones con algunas de la literatura y dio buenos resultados para el modelo construido.

Una vez que se tuvo el modelo construido y el método de solución implementado y funcionando, se buscaron los datos actuales de incidencias delictivas y el número de patrullas por cuadrantes en Ciudad de México para poder desarrollar un caso de estudio que nos permitiera probar el modelo propuesto. Se obtuvieron los datos de los cuadrantes policiales y las carpetas de investigación del portal de Datos Abiertos de la Ciudad de México. Una vez obtenidos los datos, nos dimos a la tarea de identificar los delitos de mayor incidencia y que podrían generar un mayor impacto en la demarcación seleccionada, de tal manera, que dichos puntos sean los nodos de la red y la conexión entre ellos la distancia real que existe. Así, poder probar el modelo y encontrar las rutas que pueda recorrer la patrulla dentro de la demarcación seleccionada.

Para la solución mediante el algoritmo genético tomamos una solución aleatoria con una población de 500 individuos y 50 generaciones, para la selección se tomaron a los mejores 10 individuos de cada generación para poder hacer la cruza con una probabilidad de cruce de 0.6 y un valor aleatorio de 0.8. Y para la mutación se tomó una probabilidad de 0.1 sobre cada uno de los descendientes y tener una perturbación de los valores mediante un valor aleatorio que se distribuye de forma uniforme con parámetros (0,1).

Así, con esta solución pudimos desarrollar un modelo de rutas de patrullaje aleatorias para asegurar que no fueran predictivas, asimismo, mediante la aplicación de dicho modelo en el cuadrante Álamos de la alcaldía Benito Juárez pudimos observar que efectivamente lográbamos maximizar la cobertura por parte de la policía de los puntos de mayor riesgo en cuadrantes de mayor incidencia delictiva.

Como extensión del trabajo, se podría ampliar la cantidad de cuadrantes policiales donde se puedan proponer rutas policiales, tomar

en cuenta distintos tipos de delitos para obtener más puntos de visita, considerar el uso de más de una patrulla e incluso plantear rutas dentro de un horario establecido o turnos establecidos, para esto una propuesta sería tomar los elementos esenciales del problema del agente viajero con ventanas de tiempo para poder establecer las rutas en ciertos horarios. Otro punto importante que se podría tomar para una investigación posterior sería el modificar la forma de seleccionar los elementos para la cruza y mutación dentro de nuestro algoritmo genético, ya que, como vimos en el capítulo 3 para ejemplos con una mayor cantidad de puntos o ciudades el algoritmo de recocido simulado tuvo un mejor desempeño que el algoritmo genético.

Después de hacer la revisión de la literatura y ver los pocos trabajos que existen relacionados con estrategias de optimización para enfrentar la criminalidad y los muchos trabajos de optimización combinatoria y de aplicación relacionados con las diferentes variaciones de modelos relacionados con el problema del agente viajero o el problema de ruteo, este trabajo me permitió establecer una conexión entre estos dos temas que me interesan mucho de manera que pude utilizar el TSP para desarrollar un modelo propio que me ayudara a resolver mi inquietud respecto a una forma de ayudar en el combate contra la inseguridad utilizando el parque vehicular policial ya existente y proponer una organización de tal manera que el actuar dentro de estos cuadrantes policiales establecidos por las autoridades sea la mejor posible. Asimismo, para el desarrollo del modelo y del algoritmo de solución fue indispensable cada una de las recomendaciones y aprendizajes dados en mi periodo de estudio, tanto para el modelado de la variación del TSP necesaria para encontrar las rutas como la parte lógica del armado del algoritmo genético en Python y la prueba del modelo con el caso de estudio de una demarcación de la Ciudad de México.

Bibliografía

[1] Jorge Aguirre. La aplicación de las tecnologías de información y comunicación en la rpevención comunitaria del delito: los casos de georreferenciación en monterrey, méxico. *Revista de Relaciones Internacionales, Estrategia y Seguridad*, 11, 2016.

[2] Zaida E. Alarcón Bernal, Ricardo Aceves García, and Arturo Fuentes Zenón. A conceptual model that identifies mathematical models and lean techniques for problem-solving at the different decision-making levels of service companies. *Journal of Service Science and Management*, 2021.

[3] Enrique Alba, Manuel Laguna, and Rafael Martí. Métodos evolutivos en problemas de optimización. *Revista INGENIERÍA UC*, 10(3), 2003.

[4] Béla Bollobás. *Modern Graph Theory*. Springer, 1998.

[5] Observatorio Nacional Ciudadano. Observatorio nacional ciudadano, 2021.

[6] César Cossio Guerrero and Nilse Pamela Romero Basurto. Colaboración en la elaboración del código algoritmo genético, 2021.

[7] Universidad ICESI Curso Analítica prescriptiva. Algoritmo genético completo tsp, 2020.

[8] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large -scale traveling-salesman problem. *Journal of the Operations Research. Vol. 2*, 1954.

[9] Sevda Dayioglu Gulcu and Humar Kahramanli Ornek. Solution of multiple travelling salesman problem using particle swarm optimization based algorithms. *International Journal of Intelligent Systems and Applications in Engineering*, 2019.

[10] Gobierno de la Ciudad de México. Portal de datos abiertos de la cdmx, 2021.

[11] Oficina de las Naciones Unidas contra la droga y el delito. Manual de capacitación sobre vigilancia en el espacio urbano.

[12] Abraham Duarte Muñoz, Juan José Pantrigo Fernández, and Micael Gallego Carrilo. *Metaheurísticas*. Dykinson, 2007.

[13] Erika Gineth Espinosa Téllez, Orión Sánchez Rodríguez, and Jainet Orlando Bernal. Problema del agente viajero. *Ingeciencia*, 2016.

[14] José Luis Gónzalez Velarde and Roger Z Ríos Mercado. Investigación de operaciones en acción: Aplicación del TSP en problemas de manufactura y logística. *Ingenierías*, II(4), 1999.

[15] Tao Cheng Huanfa Chen and Xinyue Ye. Designing efficient and balanced police patrol districts on an urban street network. *International Journal of Geographical Information Science*, 2018.

[16] INEGI. Encuesta nacional de victimización y percepción sobre seguridad pública (ENVIPE), 2020.

[17] INEGI. Instituto nacional de estadística y geografía (INEGI), 2021.

[18] Sonia Kefi, Nizar Rokbani, and Adel M. Alimi. Solving the traveling salesman problem using ant colony metaheuristic, a review. In *Proceedings of the 16th International Conference on Hybrid Intelligent Systems (HIS 2016)*. Springer International Publishing, 2017.

[19] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, 1992.

[20] Rafael Martí Cunquero. Algoritmos heurísticos en optimización combinatoria.

[21] Rajesh Matal, Surya Singh, and Murari Lal Mittal. Traveling salesman problem: an overview formulations, and solution approaches. *IntechOpen*, 2010.

[22] B. Min, Y. Shipin, X. Yunchen, and L. Lijuan. An improved ant colony algorithm for traveling salesman problem. In *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 1, 2019.

[23] Stefan Näher. The travelling salesman problem. In *Algorithms Unplugged*. Springer, 2011.

[24] Alfredo Olivera. Heurísticas para problemas de ruteo de vehículos, 2004.

[25] Eneko Osaba, Xin-She Yang, and Javier Del Ser. Chapter 9 - traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. In Xin-She Yang, editor, *Nature-Inspired Computation and Swarm Intelligence*, pages 135 – 164. Academic Press, 2020.

[26] Yawei Pang, Lan Zhang, Haichuan Ding, Yuguang Fang, and Shigang Chen. SPATH: Finding the safest walking path in smart cities. *IEEE Transactions on vehicular technology*, 2019.

[27] Anitha Rao, Sandeep Kumar Hegde, A Rao, K Hegde, IAnitha Rao, K Hegde, A Rao, and SK Hegde. Literature survey on travelling salesman problem using genetic algorithms. *International Journal of Advanced Research in Eduation Technology (IJARET)*, 2(1), 2015.

[28] Gerhard Reinelt. Tsplib, 2021.

[29] Mauricio G.C Resende and Ribeiro Celso C. *Optimization by GRASP*. Springer, 2016.

[30] Mauricio G.C. Resende and Celso C. Ribeiro. *Optimization by GRASP*. Springer, 2016.

[31] Linda Bibiana Rocha Medina, Elsa Cristina González La Rota, and Javier Arturo Orjuela Castro. Una revisión al estado del arte del

problema de ruteo de vehículos: Evolución histórica y métodos de solución. *Ingeniería*, 16, 2011.

[32] Katya Rodríguez Vázquez. Notas de clase, 2019.

[33] Katya Rodríguez Vázquez. Notas de clase, 2019.

[34] F. Sagasti and I. Mitroff. Operations research from the viewpoint of general systems theory. *Omega, 1*, 1973.

[35] Juan (CIDE-División de Estudios Jurídicos) Salgado Ibarra. Documento conceptual-metodológico sobre políticas públicas de seguridad ciudadana, capacidades institucionales para medir su desempeño y bases para el desarrollo de indicadores e esta materia.

[36] Linus Schrage. *Optimization modeling with Lingo*. Lindo Systems Inc., 1998.

[37] Alexander Schrijver. On the history of combinatorial optimization. In *Handbooks in OR & MS*, volume 12. Elsevier, 2005.

[38] Nico L.J. Ulder, Aarts Emile H.L., Hans-Jürgen Bandelt, Peter J.M van Laarhoven, and Erwin Pesch. Genetic local search algorithms for the trveling salesman problem.

[39] Darrell Whitley. A genetic algorithm tutorial, 1993.

[40] Yuan Yao, Zhe Peng, and Bin Xiao. Parallel hyper-heuristic algorithm for multi-objective route planning in a smart city. *IEEE Transactions on vehicular technology*, 2018.

[41] S.H. Zanakins and J.R. Evans. Heuristic 'optimization': Why, when and how to use it. *Interfaces*, 11, 1981.

Código Búsqueda Local

A continuación se presenta el código del algoritmo de búsqueda local utilizado en el capítulo 4

```
def DistanciaTotal(Solucion):
distancia = [[0,   702,  454,  842,  2396, 1196, 864,  772,
    714,  554,  2363, 2679],
[702,  0,    324,  1093, 2136, 764,  845,  764,  459,  294,
    2184, 2187],
[454,  324,  0,    1137, 2180, 798,  664,  572,  284,  338,
    2228, 2463],
[842,  1093, 1137, 0,    1616, 1857, 1706, 1614, 1421, 799,
    1521, 2021],
[2396, 2136, 2180, 1616, 0,    2900, 2844, 2752, 2464, 1842,
    95,  405],
[1196, 764,  798,  1857, 2900, 0,    396,  424,  514,  1058,
    2948, 2951],
[864,  845,  664,  1706, 2844, 396,  0,    92,   386,  1002,
    2892, 3032],
[772,  764,  572,  1614, 2752, 424,  92,   0,    305,  910,
    2800, 2951],
[714,  459,  284,  1421, 2464, 514,  386,  305,  0,    622,
    2512, 2646],
[554,  294,  338,  799,  1842, 1058, 1002, 910,  622,  0,
    1890, 2125],
[2363, 2184, 2228, 1521, 95,   2948, 2892, 2800, 2512, 1890, 0,
     500],
[2679, 2187, 2463, 2021, 405,  2951, 3032, 2951, 2646, 2125,
    500,  0]]

#print(distancia)

DistanciaTotal = 0

for i in range(0, len(Solucion)):
```

```
if i == (Solucion[i]-1):
return -1

DistanciaTotal += distancia[i][Solucion[i]-1]

DistanciaTotal += distancia[Solucion[11]][Solucion[0]-1]

return DistanciaTotal

def Perm (radio, vector):
x = []
for i in range(0, radio):
x.append(random.randint(1, len(vector)))
aux = vector[x[0]-1]
for i in range(1, len(x)):
vector[x[i-1]-1] = vector[x[i]-1]
vector[x[i]-1] = aux
return vector

N=100 # N numero de iteraciones
Cuidades = 12
ruta = []
for i in range(1,Cuidades+1):
ruta.append(i)

while DistanciaTotal(ruta) < 0:
\#    Perm(ruta)
random.shuffle(ruta)

radio=5
s0=ruta[:]
r=s0[:]
evals0 = DistanciaTotal(r)
for i in range(1, N):
s1=Perm(radio, r)
evals1 = DistanciaTotal(s1)
while evals1 < 0:
s1 =Perm(radio, r)
evals1 = DistanciaTotal(s1)
if evals1 < evals0:
s0 = s1
evals0=evals1

print (s0, evals0)
```

Código Recocido Simulado

A continuación se presenta el código del algoritmo de recocido simulado utilizado en el capítulo 4

```
import random
import math

def DistanciaTotal(Solucion):
distancia = [[0,   702,  454,  842,  2396, 1196, 864,  772,
    714,  554,  2363, 2679],
[702,  0,    324,  1093, 2136, 764,  845,  764,  459,  294,
    2184, 2187],
[454,  324,  0,    1137, 2180, 798,  664,  572,  284,  338,
    2228, 2463],
[842,  1093, 1137, 0,    1616, 1857, 1706, 1614, 1421, 799,
    1521, 2021],
[2396, 2136, 2180, 1616, 0,    2900, 2844, 2752, 2464, 1842,
    95,  405],
[1196, 764,  798,  1857, 2900, 0,    396,  424,  514,  1058,
    2948, 2951],
[864,  845,  664,  1706, 2844, 396,  0,    92,   386,  1002,
    2892, 3032],
[772,  764,  572,  1614, 2752, 424,  92,   0,    305,  910,
    2800, 2951],
[714,  459,  284,  1421, 2464, 514,  386,  305,  0,    622,
    2512, 2646],
[554,  294,  338,  799,  1842, 1058, 1002, 910,  622,  0,
    1890, 2125],
[2363, 2184, 2228, 1521, 95,   2948, 2892, 2800, 2512, 1890, 0,
       500],
[2679, 2187, 2463, 2021, 405,  2951, 3032, 2951, 2646, 2125,
    500,  0]]

DistanciaTotal = 0

for i in range(0, len(Solucion)):
```

```
if i == (Solucion[i]-1):
return -1

DistanciaTotal += distancia[i][Solucion[i]-1]

DistanciaTotal += distancia[Solucion[11]][Solucion[0]-1]

return DistanciaTotal

def Perm (radio, vector):
x = []
for i in range(0, radio):
x.append(random.randint(1, len(vector)))
aux = vector[x[0]-1]
for i in range(1, len(x)):
vector[x[i-1]-1] = vector[x[i]-1]
vector[x[i]-1] = aux
return vector

Cuidades = 12
ruta = []
for i in range(1,Cuidades+1):
ruta.append(i)

while DistanciaTotal(ruta) < 0:
#    Perm(ruta)
random.shuffle(ruta)


Ti=1000
Tf=10
c=0.9
N=100
alpha = 0.4
radio = 5

s0=ruta[:]
R=s0[:]
evals0 = DistanciaTotal(R)
T = Ti
while T > Tf:
for i in range(1, N):
s1=Perm(radio,R)
evals1 = DistanciaTotal(s1)
while evals1 < 0:
s1 =Perm(radio, R)
evals1 = DistanciaTotal(s1)
if evals1 < evals0:
```

```
s0 = s1
evals0=evals1
else:
r=random.uniform(0,1)
if r < alpha*math.exp(-1*abs(evals1 - evals0)/T):
s0 = s1
evals0=evals1
T=c*T

print(s0, evals0)
```

Código Algoritmo Genético Ejemplo

A continuación se presenta el código del algoritmo genético utilizado en el capítulo 4

[1]:
```
import numpy as np
import random
import operator
import pandas as pd
import matplotlib.pyplot as plt
from random import sample
```

Declaramos la distancia total del problema y generamos la función de evaluación

[2]:
```
def DistanciaTotal(Solucion):

distancia = [[0,   702,  454,  842,  2396, 1196,␣
↪864,  772,  714,  554,  2363, 2679],
[702,  0,    324,  1093, 2136, 764,  845,  764, ␣
↪459,  294,  2184, 2187],
[454,  324,  0,    1137, 2180, 798,  664,  572, ␣
↪284,  338,  2228, 2463],
[842,  1093, 1137, 0,    1616, 1857, 1706, 1614,␣
↪1421, 799,  1521, 2021],
```

```
        [2396, 2136, 2180, 1616, 0,    2900, 2844, 2752,␣
↪2464, 1842,  95,  405],
        [1196, 764,  798,  1857, 2900, 0,    396,  424, ␣
↪514,  1058, 2948, 2951],
        [864,  845,  664,  1706, 2844, 396,  0,    92,  ␣
↪386,  1002, 2892, 3032],
        [772,  764,  572,  1614, 2752, 424,  92,   0,   ␣
↪305,  910,  2800, 2951],
        [714,  459,  284,  1421, 2464, 514,  386,  305, ␣
↪0,    622,  2512, 2646],
        [554,  294,  338,  799,  1842, 1058, 1002, 910, ␣
↪622,  0,    1890, 2125],
        [2363, 2184, 2228, 1521, 95,   2948, 2892, 2800,␣
↪2512, 1890, 0,    500],
        [2679, 2187, 2463, 2021, 405,  2951, 3032, 2951,␣
↪2646, 2125, 500,  0]]

        DistanciaTotal = 0

        for i in range(0, len(Solucion)-1):

        DistanciaTotal +=␣
↪distancia[Solucion[i]][Solucion[i+1]]

        DistanciaTotal +=␣
↪distancia[Solucion[11]][Solucion[0]]

        return DistanciaTotal
```

Generamos la ruta inicial aleatoria

[3]:
```
        def Ruta():
        Cuidades = 12
        ruta = []
        for i in range(Cuidades):
        ruta.append(i)
```

```
        random.shuffle(ruta)
        return ruta
```

Generamos la población inicial

[4]:
```
        def PoblacionInicial(TamañoPoblacion):
        poblacion=[]
        for i in range(0,TamañoPoblacion):
        poblacion.append(Ruta())
        return poblacion
```

Se realiza la selección de la población

[5]:
```
        def Seleccion(Poblacion,elit):
        l=len(Poblacion)
        selectionResults = []
        df=pd.
↪DataFrame(list(range(l)),columns=['Individuo'])
        df["Ruta"] = Poblacion
        df["Aptitud"] = [DistanciaTotal(x) for x in␣
↪Poblacion]
        df['Individuo']=list(range(l))#Creo el dataframe
        df=df.sort_values('Aptitud').reset_index().
↪drop('index', axis=1) #Reacomodar
        df['cum_sum'] = df.Aptitud.cumsum() #Selección de␣
↪los mejores
        df['cum_perc'] = 100*df.cum_sum/df.Aptitud.sum()

        for i in range(0, elit):
        selectionResults.append(df.iloc[i])
        for i in range(0, len(df) - elit):
        pick = 100*random.random()
        for i in range(0, len(df)):
        if pick <= df.iat[i,4]:
```

```
        selectionResults.append(df.iloc[i])
        break
        df=df.drop('cum_sum', axis=1).drop('cum_perc',␣
↪axis=1)[:]

        l1=list(range(0,l))  #Se realizan permutaciones␣
↪sobre los índices
        l2=list(range(0,l))
        random.shuffle(l1)
        random.shuffle(l2)
        I=[]   #Lista de índices seleccionados
        for i in range(0, len(Poblacion)):
        if df['Aptitud'][l1[i]] < df['Aptitud'][l2[i]]:
        I.append(l1[i])
        else:
        I.append(l2[i])
        S1=df.iloc[I].sort_values('Aptitud').
↪reset_index().drop('index', axis=1)[:] #Reacomodar
        S1['Individuo']=list(range(l))

        return S1[0:elit] #Individuos ganadores en␣
↪formato dataframe
```

Función de cruza entre dos padres para crear al hijo

[6]:

```
        def Cruza(S,I):
        P=S['Ruta']
        F=[]
        l=len(P)
        for j in range(int(I/2)):  #Para generar los␣
↪padres con una probabilidad de cruce pc y un aleatorio␣
↪al
        pc = 0.6 #Probabilidad de cruce
        al = 0.8
        while al > pc:
        s=sample(range(l),2)
```

```
        al=np.random.uniform(0,1,1)[0]
        F.append(s)

        R=len(P[0])
        c1=int(R/3)       #Puntos de corte y de casi del␣
↪mismo tamaño
        c2=int((R-int(R/3))/2)+c1

        df=pd.DataFrame()
        Hijos=[]
        D=[]
        for i in range(int(I/2)):

        h1=P[F[i][0]][0:c1]+P[F[i][1]][c1:
↪c2]+P[F[i][0]][c2:R] #Estos son los hijos
        h2=P[F[i][1]][0:c1]+P[F[i][0]][c1:
↪c2]+P[F[i][1]][c2:R]

        e1=sorted(enumerate(h1),key=lambda x:x[1])
        e2=sorted(enumerate(h2),key=lambda x:x[1])
        E1=[e1[i][0] for i in range(len(e1)-1) if␣
↪e1[i][1]==e1[i+1][1]] #Indices de números repetidos␣
↪solo uno
        E2=[e2[i][0] for i in range(len(e2)-1) if␣
↪e2[i][1]==e2[i+1][1]] #Indices de números repetidos␣
↪solo uno
        EF1=[x for x in list(range(12)) if x not in h1]␣
↪#Elementos faltantes
        EF2=[x for x in list(range(12)) if x not in h2]␣
↪#Elementos faltantes
        H1=h1[:]
        H2=h2[:]

        for m in range(len(E1)):
        H1[E1[m]]=EF1[m]   #Cambio de elementos repetidos␣
↪por elementos faltantes para el hijo1

        for m in range(len(E2)):
```

```
        H2[E2[m]]=EF2[m]    #Cambio de elementos repetidos␣
↪por elementos faltantes para el hijo1

        Hijos.append(H1)
        Hijos.append(H2)
        D.append(DistanciaTotal(H1))
        D.append(DistanciaTotal(H2))

        df=pd.
↪DataFrame(list(range(I)),columns=['Individuo'])
        df["Ruta"] = Hijos
        df["Aptitud"] = D
        return df
```

[7]:
```
        def Mutacion(C):
        l=len(C)
        L=len(C['Ruta'][0])
        for i in range(l):
        pm=0.1
        pg=np.random.uniform(0,1,1)[0]
        if pg<pm:
        s=sample(range(L),2)
        C['Ruta'][i][C['Ruta'][i].index(s[0])]=s[1]
        C['Ruta'][i][C['Ruta'][i].index(s[1])]=s[0]
        return C
```

Se calcula la siguiente generación

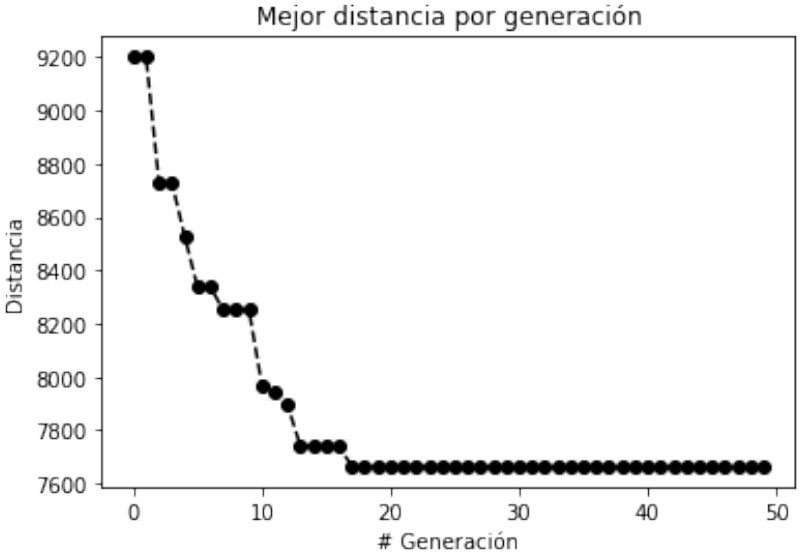
[8]:
```
        def SiguienteGeneracion(P, I, e):
        S=Seleccion(P,e)
        C=Cruza(S,I)
        M=Mutacion(C)
        return M
```

Algoritmo genético aplicando todas la funciones anteriores

[9]:
```
        def GATSP(Individuos, NGene, elit): #NGene es el␣
↪número de generaciones que se quieren

        SG=PoblacionInicial(Individuos)
        y=[]
        r=[]
        for i in range(NGene):
        SG = SiguienteGeneracion(SG, Individuos, elit)
        M=SG.sort_values('Aptitud')
        y.append(M.iloc[0]['Aptitud'])
        r.append((M.iloc[0]['Ruta'],M.iloc[0]['Aptitud']))
        SG=list(SG['Ruta'])
        plt.plot(np.arange(0, NGene, 1),y, '--ok')
        plt.title('Mejor distancia por generación')
        plt.xlabel('# Generación')
        plt.ylabel('Distancia')
        plt.show()
        return r
```

[51]:
```
        R = GATSP(Individuos=500,NGene=50,elit=6)
```



[52]:
```
len(R)
```

[52]:
```
50
```

[53]:
```
R[len(R)-1]
```

[53]:
```
([8, 2, 1, 9, 11, 4, 10, 3, 0, 7, 6, 5], 7664)
```

[ ]:

Matriz de distancias caso de estudio

A continuación se presenta la matriz con los datos utilizados en el caso de estudio

Tabla 1: Matriz Parte 1

	PC	N1	VP1	VP2	VP3	VP4	AM1	AM2	AM3	AM4
PC	0	1088.153	718.653	372.79	498.388	736.109	571.158	583.394	718.653	769.202
N1	326.223	0	1019.6	698.878	295.223	473.767	494.989	380.23	1019.6	1070.15
VP1	520.702	806.002	0	893.357	489.702	721.677	689.468	574.709	0.1314	50.681
VP2	1130.711	1197.625	923.838	0	559.283	607.406	632.053	644.29	923.838	974.388
VP3	697.276	764.19	1160.29	954.446	0	580.119	199.611	84.852	1160.299	1210.849
VP4	736.468	537.972	946.048	1048.805	645.151	0	774.275	665.118	946.048	996.598
AM1	1050.616	1117.53	960.252	988.177	904.132	763.75	0	562.014	960.252	1010.801
AM2	1218.668	1285.582	1128.303	1156.228	1072.184	654.884	516.034	0	1128.303	1178.853
AM3	520.589	805.89	0.0185	893.244	489.589	721.677	689.355	574.596	0	50.568
AM4	470.517	755.817	1163.894	843.172	439.517	671.127	639.282	524.524	1163.894	0
AM5	448.186	733.487	1141.563	820.841	417.187	649.142	616.952	502.194	1141.56	1192.113
AM6	604.438	278.338	1133.602	977.092	573.438	641.378	773.203	658.445	1133.602	1184.151
AM7	714.967	781.881	1177.99	972.137	17.5367	597.505	217.301	102.543	1177.99	1228.54
AM8	734.594	801.508	1197.617	991.765	588.11	527.848	236.929	128.063	1197.617	1248.167
AM9	296.589	581.889	989.966	669.243	265.589	497.708	465.354	350.596	989.966	1040.515
AM10	369.925	43.825	1063.301	742.579	338.925	451.974	538.6	423.932	1063.3	1113.851
AM11	386.536	60.436	915.7	759.19	355.536	423.398	555.301	440.543	915.7	966.249
AM12	683.128	357.028	765.104	1055.783	652.128	272.847	851.894	737.135	765.104	815.654
AM13	686.424	360.324	768.4	1059.079	655.424	276.056	803.105	694.239	768.4	818.95
AM14	852.895	919.809	1327.886	1165.233	761.578	172.7793	890.702	781.837	1327.886	1378.435

Tabla 2: Matriz Parte 2

	AM5	AM6	AM7	AM8	AM9	AM10	AM11	AM12	AM13	AM14
PC	791.187	1005.731	481.002	711.536	942.621	1044.344	1223.711	1226.87	776.479	978.073
N1	1092.135	874.786	277.83	508.372	1243.569	1174.983	1092.766	452.867	573.315	452.867
VP1	72.666	1069.4625	472.316	702.851	224.099	1369.462	1287.245	896.148	767.793	647.345
VP2	996.373	1210.917	541.897	772.431	1147.807	1249.53	1428.897	1287.771	837.374	1038.968
VP3	1232.834	1027.452	533.406	443.406	1384.268	1327.649	1245.432	854.336	403.939	605.533
VP4	1018.583	801.234	627.765	537.483	1170.017	1101.431	1019.214	628.118	443.131	172.838
AM1	1032.786	1247.33	886.746	796.464	1184.22	1285.943	1465.31	1207.676	757.279	958.873
AM2	1200.838	1415.382	1054.798	964.516	1352.272	1453.995	1633.362	1375.728	925.331	1126.925
AM3	72.553	1069.152	472.204	702.738	223.987	1369.349	1287.132	896.036	767.681	647.233
AM4	22.481	1019.08	422.131	652.665	173.914	1319.277	1237.06	845.963	717.608	597.16
AM5	0	996.749	399.801	630.335	151.584	1296.94	1214.73	502.194	695.278	574.83
AM6	1206.136	0	556.529	786.586	1357.57	1288.985	217.933	368.484	851.529	731.081
AM7	1250.525	1045.143	0	460.814	1401.958	1345.34	1263.123	872.027	421.63	623.224
AM8	1270.152	1064.771	570.724	0	1421.586	1364.968	1282.751	891.654	441.257	642.851
AM9	1062.5	845.152	248.203	478.737	0	1145.349	1063.132	672.035	543.68	423.232
AM10	1135.836	918.488	321.539	552.073	1287.27	0	1136.468	745.371	617.016	496.568
AM11	988.234	770.886	338.15	568.684	1139.668	1071.083	0	150.581	633.627	513.17
AM12	837.639	620.291	634.743	865.277	989.073	920.488	838.271	0	930.219	809.772
AM13	840.935	623.587	638.038	566.312	992.369	923.784	841.567	450.47	0	201.667
AM14	1400.42	1183.072	744.192	653.91	1551.854	1483.269	1401.052	1009.955	559.558	0

Matriz de distancias del código de Python

La matriz de distancias del caso de estudio

[2]:
```
        Dist = [[0, 1088.153, 718.653, 372.79, 498.388,␣
↪736.109, 571.158, 583.394, 718.653, 769.202, 791.187,␣
↪1005.731, 481.002, 711.536, 942.621, 1044.344, 1223.
↪711, 1226.87, 776.479, 978.073],
        [326.223, 0, 1019.6, 698.878, 295.223, 473.767,␣
↪494.989, 380.23, 1019.6, 1070.15, 1092.135, 874.786,␣
↪277.83, 508.372, 1243.569, 1174.983, 1092.766, 452.867,␣
↪573.315, 452.867],
        [520.702, 806.002, 0, 893.357, 489.702, 721.677,␣
↪689.468, 574.709, 0.1314, 50.681, 72.666, 1069.4625,␣
↪472.316, 702.851, 224.099, 1369.462, 1287.245, 896.148,␣
↪767.793, 647.345],
        [1130.711, 1197.625, 923.838, 0, 559.283, 607.
↪406, 632.053, 644.29, 923.838, 974.388, 996.373, 1210.
↪917, 541.897, 772.431, 1147.807, 1249.53, 1428.897,␣
↪1287.771, 837.374, 1038.968],
        [697.276, 764.19, 1160.29, 954.446, 0, 580.119,␣
↪199.611, 84.852, 1160.299, 1210.849, 1232.834, 1027.
↪452, 533.406, 443.406, 1384.268, 1327.649, 1245.432,␣
↪854.336, 403.939, 605.533],
```

```
       [736.468, 537.972, 946.048, 1048.805, 645.151, 0,␣
↪774.275, 665.118, 946.048, 996.598, 1018.583, 801.234,␣
↪627.765, 537.483, 1170.017, 1101.431, 1019.214, 628.
↪118, 443.131, 172.838],
       [1050.616, 1117.53, 960.252, 988.177, 904.132,␣
↪763.75, 0, 562.014, 960.252, 1010.801, 1032.786, 1247.
↪33, 886.746, 796.464, 1184.22, 1285.943, 1465.31, 1207.
↪676, 757.279, 958.873],
       [1218.668, 1285.582, 1128.303, 1156.228, 1072.
↪184, 654.884, 516.034, 0, 1128.303, 1178.853, 1200.838,␣
↪1415.382, 1054.798, 964.516, 1352.272, 1453.995, 1633.
↪362, 1375.728, 925.331, 1126.925],
       [520.589, 805.89, 0.0185, 893.244, 489.589, 721.
↪677, 689.355, 574.596, 0, 50.568, 72.553, 1069.152, 472.
↪204, 702.738, 223.987, 1369.349, 1287.132, 896.036, 767.
↪681, 647.233],
       [470.517, 755.817, 1163.894, 843.172, 439.517,␣
↪671.127, 639.282, 524.524, 1163.894, 0, 22.481, 1019.
↪08, 422.131, 652.665, 173.914, 1319.277, 1237.06, 845.
↪963, 717.608, 597.16],
       [448.186, 733.487, 1141.563, 820.841, 417.187,␣
↪649.142, 616.952, 502.194, 1141.56, 1192.113, 0, 996.
↪749, 399.801, 630.335, 151.584, 1296.94, 1214.73, 502.
↪194, 695.278, 574.83],
       [604.438, 278.338, 1133.602, 977.092, 573.438,␣
↪641.378, 773.203, 658.445, 1133.602, 1184.151, 1206.
↪136, 0, 556.529, 786.586, 1357.57, 1288.985, 217.933,␣
↪368.484, 851.529, 731.081],
       [714.967, 781.881, 1177.99, 972.137, 17.5367, 597.
↪505, 217.301, 102.543, 1177.99, 1228.54, 1250.525, 1045.
↪143, 0, 460.814, 1401.958, 1345.34, 1263.123, 872.027,␣
↪421.63, 623.224],
       [734.594, 801.508, 1197.617, 991.765, 588.11, 527.
↪848, 236.929, 128.063, 1197.617, 1248.167, 1270.152,␣
↪1064.771, 570.724, 0, 1421.586, 1364.968, 1282.751, 891.
↪654, 441.257, 642.851],
```

```
       [296.589, 581.889, 989.966, 669.243, 265.589, 497.
↪708, 465.354, 350.596, 989.966, 1040.515, 1062.5, 845.
↪152, 248.203, 478.737, 0, 1145.349, 1063.132, 672.035,␣
↪543.68, 423.232],
       [369.925, 43.825, 1063.301, 742.579, 338.925, 451.
↪974, 538.6, 423.932, 1063.3, 1113.851, 1135.836, 918.
↪488, 321.539, 552.073, 1287.27, 0, 1136.468, 745.371,␣
↪617.016, 496.568],
       [386.536, 60.436, 915.7, 759.19, 355.536, 423.
↪398, 555.301, 440.543, 915.7, 966.249, 988.234, 770.
↪886, 338.15, 568.684, 1139.668, 1071.083, 0, 150.581,␣
↪633.627, 513.17],
       [683.128, 357.028, 765.104, 1055.783, 652.128,␣
↪272.847, 851.894, 737.135, 765.104, 815.654, 837.639,␣
↪620.291, 634.743, 865.277, 989.073, 920.488, 838.271,␣
↪0, 930.219, 809.772],
       [686.424, 360.324, 768.4, 1059.079, 655.424, 276.
↪056, 803.105, 694.239,        768.4,        818.
↪95,        840.935, 623.587, 638.038, 566.312, 992.369,␣
↪923.784, 841.567, 450.47, 0, 201.667],
       [852.895, 919.809, 1327.886, 1165.233, 761.578,␣
↪172.7793, 890.702, 781.837, 1327.886, 1378.435, 1400.
↪42, 1183.072, 744.192, 653.91, 1551.854, 1483.
↪269,        1401.052, 1009.955, 559.558, 0]]
```