Facultad de Ciencias Naturales y Matemáticas

Algoritmo Concorde: Análisis y aplicación al Problema del Agente Viajero Simétrico

Proyecto Integrador

Previo a la obtención del título de:

Matemático

Presentado por:

Cristina Lorenti

Melissa Quimis

ESPOL

Guayaquil-Ecuador

2022

Dedicatoria

A Yazmin y Aquiles, mis amados padres por ser mi ejemplo a seguir cada día.

Cristina Lorenti

A quienes nunca dudaron que podría lograrlo, y en su confianza me tuvieron en sus oraciones. A mi Olayita y a mi mami, por su fidelidad y amor. Mi papá, hermanos, sobrinas y toda mi familia. Y a Jhon que hace un año también creyó en mí.

Melissa Quimis

Agradecimientos

A Dios,
a mis padres y hermanos,
a mi hermana de otra madre Melissa,
a JuanJo, Marco, Ale y Jhixon,
al Dr. Xavier Cabezas y la Dra. Marchan,
a mis profesores de carrera que me guiaron durante este largo camino.

Cristina Lorenti

A Dios que me amó primero,
a mi mami y mi familia por el apoyo,
a Cristina, mi Matemática favorita,
a mi team de cualquier materia,
al Dr. Cabezas y la Dra. Elimar
a cada profesor que me ayudó a crecer,
a Margarita, por motivarme a quedar en la carrera.

Melissa Quimis

Declaración

"Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Cristina Marisol Lorenti Zambrano y Melissa Selena Quimis Soledispa damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Cristina Marisol Lorenti Zambrano

Melissa Selena Quimis Soledispa

Evaluadores

Luz Elimar Marchan M. Ph.D.
PROFESORA DE LA
MATERIA

Xavier Cabezas G. Ph.D.
PROFESOR TUTOR

Resumen

El continuo estudio del problema del agente viajero se ha convertido en un objetivo de vital importancia, que resulta de alta complejidad. Se han desarrollado distintos métodos o algoritmos que permiten obtener una solución óptima o encontrarse cerca de su entorno. Cook y otros, desarrollaron el algoritmo Concorde, un método exacto, que es considerado el más eficaz para instancias grandes hasta la actualidad. En este trabajo se presenta un esquema de su estructura, aplicación y comparación, frente a con otros métodos. Para validar su exactitud se probaron distintas instancias encontradas en la literatura, incluyendo un ejemplo propuesto de provincias de Ecuador, utilizando la interfaz del algoritmo que está disponible para efectos académicos. Además, se usó Microsoft Excel y Wolfram Mathematica para obtener la matriz distancia de las coordenadas que se estudiaron. También se usó el ejecutor de Python el cual generó las soluciones óptimas que se compararon con los distintos métodos, como el método del vecino más cercano y LinKer. Finalmente, se obtuvo que el método del vecino más cercano no es tan eficaz como se esperaba, mientras que el método Lin-Ker permitió obtener la solución óptima para instancias bastante altas. Por otra parte, se obtuvo que el algoritmo Concorde que genera la solución óptima para todas las instancias propuestas, toma un tiempo considerable corto que permite optimizar el recorrido del problema propuesto. Así, se concluye que el algoritmo Concorde presenta un nivel de eficiencia y exactitud en el desarrollo de problemas simétricos del tipo agente viajero.

Palabras claves: Problema del Agente Viajero Simétrico, algoritmo Concorde.

Abstract

The continuous study of the traveling salesman problem has become a very important approach, which is highly complex. Different methods or algorithms have been developed that allow one to obtain an optimal solution or to find it close to its environment. Cook and others developed the Concorde algorithm, an exact method, which is considered the most efficient for large instances to date. This paper presents an outline of its structure, application, and comparison with other methods. To validate its accuracy, different instances found in the literature were tested, including a proposed example from Ecuadorian provinces, using the algorithm interface that is available for academic purposes. In addition, Microsoft Excel and Wolfram Mathematica were used to obtain the distance matrix of the coordinates that were studied. Python was also used, which generated the optimal solutions that were compared with other methods, such as the nearest neighbor method and LinKer. Finally, it was found that the nearest neighbor method is not as efficient as expected, while the Lin-Ker method allowed obtaining the optimal solution for quite high instances. On the other hand, it was obtained that the Concorde algorithm that generates the optimal solution for all the proposed instances, takes a considerable short time that allows optimizing the path of the proposed problem. Thus, it is concluded that the Concorde algorithm presents a level of efficiency and accuracy in the development of symmetric traveling salesman problem.

Key Words: Symmetric Traveling Salesman Problem, Concorde algorithm.

Índice de figuras

Índice de cuadros

Índice general

Capítulo 1

Introducción

1.1. Descripción del problema

El estudio del Problema del Agente Viajero, o TSP por sus siglas en inglés, busca encontrar la ruta más corta visitando cada nodo y regresando a su punto de partida. Este problema se remonta a los años 1800s en ejemplos de viajes a través de Alemania y Suiza, como el proceso de entregar cartas, donde existía gran dificultad en la movilidad y poco conocimiento de rutas, véase (Cook, 2012). Pero no fue hasta los años 1930s que se planteó por primera vez este problema por matemáticos en Viena y Harvard, aunque su nombre se le atribuye al matemático Hassler Whitney. A lo largo de los años es posible reconocer que el problema del agente viajero se ha convertido en un enfoque de vital importancia, siendo considerado uno de los más estudiados, que acorde a la búsqueda de soluciones que satisfagan las aplicaciones reales del problema, su desarrollo resulta de alta complejidad. En dicho sentido se han desarrollado diferentes alternativas que han resultado de bajo o alto impacto en la solución de estos problemas, con el objetivo de obtener soluciones confiables y eficaces.

Considerando lo anterior, en los 90s, Cook y otros desarrollaron el algoritmo Concorde, que ha sido propuesto como el solucionador más eficaz de los problemas del tipo agente viajero que se han desarrollado hasta la actualidad, véase (Cook, 2012). En 2006, ellos estudiaron un problema de diseño de microchip considerando 85,900 nodos y lograron obtener un recorrido óptimo. Además, este algoritmo propone que, para muchas instancias con millones de ciudades, se garantiza que la solución contenga un $2-3\,\%$ del recorrido óptimo, véase (Cook, 2015). A pesar de ello, la falta de conocimiento de este algoritmo y de su funcionalidad ha limitado la generación de nuevos resultados, incluyendo la deficiente optimización de recursos en diferentes situaciones.

1.2. Justificación del problema

En la actualidad, el problema del agente viajero resulta de suma importancia, dado que constantemente involucra un problema social de gran magnitud, como envío de mercaderías, traslado de pasajeros (turismo, agencias de viajes, expresos laborales

y/o escolares), entre otros, véase (Penna, 2013). Por lo que, se puede decir, que este problema impacta en muchas áreas de estudio en las que se pueda evidenciar que la situación involucra una visita a varios puntos (nodos) considerando un costo entre ellos, ya sea de distancia, tiempo o dinero.

Cabe mencionar que existen varias maneras para dar solución a todas las diferentes situaciones, como hacer una lista de todas las soluciones posibles, calcular distancia, costos o tiempo, hacer una comparación y elegir la más conveniente, u otros; pero en instancias sumamente grandes esto nos tomaría toda la vida. Por ello, se plantean otras soluciones como lo son los métodos exactos o también conocidos como algoritmos óptimos, cuyo principal propósito es acelerar la búsqueda descartando muchas posibles soluciones y hallando la óptima (Penna, 2013).

Así mismo, el presente proyecto se enfoca en las heurísticas, que es otra alternativa de solución, las cuales brindan soluciones en tiempos de cómputo muy cortos, sin garantizar que la solución sea única cuando el nodo de inicio no es fijo. Este problema resulta ser uno de los más estudiados en matemáticas computacionales, su continuo estudio radica en que todavía no se conoce un método de solución efectivo para el caso general. Es por ello, que se busca analizar y dar a conocer un algoritmo recientemente aplicado al TSP, el mismo que ha sido catalogado como la mejor alternativa de solución de dicho problema para instancias grandes, que existe actualmente.

1.3. Objetivos

1.3.1. Objetivo General

Visualizar las propiedades del algoritmo Concorde para resolver el Problema del Agente Viajero Simétrico mediante su difusión y aplicación experimental en instancias académicas.

1.3.2. Objetivos Específicos

- Examinar la literatura enfocada a la resolución del Problema del Agente Viajero Simétrico a través de un estudio cronológico de su desarrollo hasta la actualidad.
- Analizar la literatura del algoritmo de Concorde aplicado al Problema del Agente Viajero Simétrico para la distinción del funcionamiento de su algoritmo.
- Aplicar el algoritmo Concorde en una instancia real para verificar sus fortalezas respecto a otros propuestos en la literatura.

1.4. Marco teórico

1.4.1. Introducción y estado del arte

Durante muchos años la búsqueda de la ruta más corta se ha convertido en un problema de constante investigación, inicialmente con el propósito de facilitar la guía de los mensajeros que recorrían diferentes ciudades para cumplir su objetivo, como lo define Cook (2012). Este enfoque de constante investigación se ha consolidado como el conocido problema del agente viajero que así mismo ha sido identificado en diferentes áreas como la medicina, ingeniería, logística y otras, en las cuales se requiere la búsqueda de información para resultados efectivos (Penna, 2013).

El problema del agente viajero (TSP), menciona que dado un conjunto de ciudades con un costo de viaje entre cada una, busque encontrar la vía más barata para visitar todas las ciudades y retornar al punto de partida. La "vía"de visitar las ciudades es simplemente el orden en la que cada ciudad es visitada; el cual es llamado *tour* o *circuito* de las ciudades (David L. Applegate, 2006). De este modo, resulta imprescindible el desarrollo de nuevos métodos de solución y técnicas para su comprensión (Cook, 2022).

Muchos investigadores han generado nuevos resultados en relación al problema mencionado, véase (Cook, 2012). Sin embargo, dado que existe un déficit de conocimiento de algoritmos avanzados que puedan ser aplicados, la solución del problema continúa alejada de la realidad.

De este modo, la investigación propuesta invita a generar un estudio detallado de uno de los algoritmos conocido por los investigadores como el más rápido y eficiente para el agente viajero que se ha desarrollado hasta la actualidad, el algoritmo Concorde (Cook, 2022). Esto se basa en las diferentes investigaciones que se generaron para dar solución al problema en mención o la mejora de resultados previos según sus respectivos algoritmos.

En el año 2006 se aplica el algoritmo denominado Concorde para desarrollar el TSP en 85.900 ciudades o nodos, donde se busca modelar el movimiento de un láser de un sitio a otro. Cabe mencionar que aunque la cantidad de nodos fue un gran reto que se logra desarrollar mediante el uso del algoritmo Concorde, resulta curioso notar que en el año 2001, aplicando el mismo algoritmo se desarrolló la solución para un problema de menor cantidad de nodos con mayor dificultad de planteamiento, véase (Cook, 2012).

Concorde se basa en el uso de un solucionador de programación lineal donde se desarrolla un proceso de búsqueda de cortes con la intención de reducir el problema. En dicha reducción, Concorde genera las restricciones del espacio de búsqueda a partir de la aplicación de los algoritmos de ramificación, véase (Cook, 2009). El lenguaje de programación que este programa usa es ANSI C, y cualquier investigador

académico puede acceder a ello. Cabe reconocer que el Concorde ha permitido adquirir 106 soluciones óptimas de las 110 que constaban en la librería de este problema (Cook, 2022).

De forma considerable el algoritmo Concorde comienza a adquirir una importancia renombrable a nivel de eficiencia y exactitud en el desarrollo de problemas simétricos del tipo agente viajero. Razón por la cual, varios investigadores en su aplicación y análisis lo comienzan a denominar como el algoritmo más efectivo y rápido para dichos problemas, véase (Cook, 2022). Se denomina al algoritmo Concorde uno del tipo complejo de bifurcación y corte computacional, donde la bifurcación permite la réplica del proceso inicial en un tamaño más reducido para obtener su solución, véase (Academic, 2022). Además, el algoritmo mencionado está combinado con los mecanismo heurísticos.

La importancia de este algoritmo radica en su adaptabilidad a los problemas de TSP simetrico, incluyendo las mejoras que se han llevado a cabo a través del mismo, permitiendo abarcar otro tipo de problemas. De forma general, no limita sus funciones y no especifica su uso a un solo tipo de solución, se conoce que incluye más de 700 funciones y que las mismas son seguras a nivel computacional para la creación de subprocesos en entornos paralelos a la programación original, véase (Cook, 2022).

1.4.2. Programación Entera

Existen dos tipos de problemas de programación lineal entera, los mismos que se describen a continuación:

- Problema de programación lineal entera puro es de la forma:

$$\begin{aligned} \max \quad & cx \\ \text{sujeto a} \quad & Ax \leq b \\ & x \geq 0 \qquad \text{integral,} \end{aligned}$$

  donde, los datos casi siempre racionales, son el vector fila $c = (c_1, \ldots, c_n)$, la matriz $m \times n$ $A = (a_{ij})$, y el vector columna $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$.

  Y el vector columna $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$, tiene las variables a optimizar. Un n-vector x se llama integral cuando $x \in \mathbb{Z}^n$. (Conforti M., 2014)

- Problema de programación lineal entera mixta que es de la forma:

$$\begin{aligned} \max \quad & cx + hy \\ \text{sujeto a} \quad & Ax + Gy \leq b \\ & x \geq 0 \qquad \text{integral} \\ & y \geq 0, \end{aligned}$$

donde, los datos casi siempre racionales, son los vectores filas $c = (c_1, \ldots, c_n)$, $h = (h_1, \ldots, h_n)$ las matrices $m \times n$ $A = (a_{ij})$, $m \times p$ $G = (g_{ij})$ y el vector columna $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$. Y los vectores columnas $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$, $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$ tienen las variables a optimizar. Algo a tener en cuenta es que las variables x_j deben ser enteros no negativos, mientras que las variables y_j pueden tomar cualquier valor real no negativo. Por beneficio nos referiremos a los programas lineales enteros mixtos como programas enteros. (Conforti M., 2014)

1.4.3. Optimización combinatoria

La optimización combinatoria es un campo relacionado al área de la combinatoria y la informática teórica, que permite desarrollar la solución de un problema de optimización discreto bajo técnicas combinatorias. En otro caso, un problema de optimización discreta tiene como objetivo hallar la mejor solución del problema propuesto bajo un conjunto finito de posibilidades (Conforti M., 2014).

En la actualidad, la optimización combinatoria hace referencia a la mejora de algoritmos o métodos que se utilizan para resolver los problemas involucrados, aplicando en su desarrollo métodos matemáticos que permitan obtener los resultados esperados. Inicialmente, este proceso solamente se enfocaba en el desarrollo de problemas relacionados a teoría de grafos.

Yepes (2014) señala que "Los problemas de optimización en los que las variables de decisión son enteras, es decir, donde el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales, reciben el nombre de problemas de optimización combinatoria". Con esta definición podemos introducir al problema del agente viajero como un problema de optimización combinatoria, como se puede ver en su formulación matemática.

1.4.4. Historia del Problema del Agente Viajero

Lo que se encuentra a continuación será una breve reseña de la historia del problema del agente viajero, la mayor parte de la información ha sido tomada de, In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation de

Cook (2012).

Inicialmente, antes de considerar el desarrollo matemático y computacional del problema del agente viajero, el proceso de descubrimiento de este tipo de problema radicó en la historia del vendedor que buscaba minimizar los costos que invertía en su ruta y más adelante, se denotó a través del repartidor de cartas y su desafío para encontrar una ruta que le facilite su labor. Alrededor del año 1832 el problema comienza a ser mencionado típicamente como "la necesidad de encontrar una buena ruta". Sin ningún método matemático, las dudas alrededor de la búsqueda de tours por diferentes ciudades al menor costo posible comienzan a surgir en algunas personas. Sin embargo, los investigadores de aquella época prefirieron por mucho tiempo evadir el tema, dado que no generaba gran conmoción a nivel científico.

Más adelante, Euler y Hamilton dan un giro a la investigación de este tipo de problemas presentando asuntos muy relacionados al agente viajero como lo son:

1. Los puentes de Königsberg, donde dados siete puentes de dicha ciudad, hoy conocida como Kaliningrado, se buscaba verificar la existencia de un camino que pase por cada uno de los puentes retornando al punto de partida sin repetir ninguno, mayor información se puede encontrar en (Núñez Valdés Juan, 2004).

2. La ruta del caballo en ajedrez, que se enfoca en los movimientos de la pieza del caballo en el tablero de ajedrez, tal que pase por todos los cuadros una sóla vez, desde un punto inicial que retorne al mismo. Claramente se debe considerar el tipo de movimiento en "L"que ejerce esta pieza. En (Cook, 2012) se puede adquirir mayor información.

3. *The icosian game*, desarrollado con base en los ciclos hamiltonianos, donde dado un grafo es posible encontrar un camino cerrado simple que pasa por cada vértice del mismo una sola vez (Harris John M. et al., 2008). A partir de ello, el juego consistía en encontrar un ciclo hamiltoniano pasando por las aristas del dodecaedro que forma el juego, cruzando cada vértice una sola vez hasta retornar al origen. Un mayor detalle de este juego se puede encontrar en (Rouse Ball W.W., 1987).

4. El dodecaedro viajero, que similar al "icosian game" recrea un dodecaedro casi plano en un tablero de madera simulando el mundo donde cada vértice es un lugar distinto. Con el objetivo de recorrer los lugares una sóla vez, partiendo de un punto y finalizando en el mismo, el libro (Cook, 2012) explica a mayor detalle el juego.

Todo lo anterior, incita a la comunidad científica a percibir el problema del agente viajero desde otra perspectiva, donde las matemáticas resultaban necesarias para el desarrollo de la optimización.

En ese sentido, la magnitud del problema comenzó a extenderse, y sus aplicaciones comenzaron a surgir con rapidez. En 1940, Mahalanobis buscaba reducir los

costos de unas tierras de las granjas de Bengal, su mayor inconveniente se presentaba a través de la cantidad de terrenos y espacios que se debían cubrir, por lo cual optó por verificar el lugar dividiendo en zonas la granja total, tomando en consideración el tiempo empleado en la movilización del personal que se veía involucrado y los equipos necesarios entre las localidades de la granja que se tomaban por muestra. Hasta dicho punto, la aplicación del agente viajero se veía evidenciada en la búsqueda de rutas eficientes para optimizar dichos recursos.

Constantes investigaciones, aplicaciones y descubrimientos se fueron llevando a cabo en relación al problema. Finalmente, en 1954, Datnzing, Fulkerson y Johnson logran resolver el problema del agente viajero con 49 nodos o ciudades, mediante su formulación con programación lineal. Un hito que marcó la investigación y desarrollo de las soluciones para este tipo de problemas. A partir de ello, se pueden recalcar los siguientes hechos de interés:

- 1962: Procter and Gamble propone el concurso "Help car 54", que buscaba hallar la ruta más corta entre un grupo de 33 nodos o ciudades.
- 1977: Martin Grötschel desarrolla en su tesis doctoral una solución para optimizar el recorrido entre 120 ciudades o nodos.
- 1987: Padberg y Rinaldi optimizan el problema del agente viajero simétrico para 532 nodos, el mismo año Grötschel desarrolló otros resultados sorprendentes para un mayor número de nodos.
- 1998: Se optimiza un recorrido para 13.509 ciudades estadounidenses.
- 2004: Se desarrolla el problema del agente viajero para 24.978 ciudades de Suecia.

De manera sorprendente para la comunidad científica, en 2006, se aplica un nuevo algoritmo que logra resolver el problema para 85.900 ciudades. Donde se puede evidenciar que el TSP ha crecido en gran magnitud y sus soluciones continúan siendo un reto para los investigadores. Después de sus aplicaciones iniciales que no generaban interés, se convirtió en uno de los problemas con mayor fama que está enfocado a la optimización. Cuya base fundamental es recorrer n-ciudaddes, empezando y terminando en el mismo origen trasladándose por cada lugar una sola ocasión. A continuación, se presenta una tabla que resume algunos récords históricos para la comunidad científica en la solución del TSP:

Año	Autores	Ciudades
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M. Karp	64
1975	P.M. Camerini, L. Fratta, F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder and M. W. Padberg	318
1987	M. Padberg and G. Rinaldi	532
1987	M. Grötschel and O. Holland	666
1987	M. Padberg and G. Rinaldi	2392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook	18512
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24978
2004	W. Cook, Espinoza and Goycoolea	33810
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook	85900

Tabla 1.1. Récords históricos en la cantidad de nodos para la resolución del TSP. Obtenido de (García, 2014)

1.4.5. Formulación Matemática

Dado que el objetivo es encontrar el recorrido óptimo, se define aquel que minimice el costo total. A este problema de programación entera se lo define como

$$x_j = \begin{cases} 1 & \text{si la ruta j fue seleccionada,} \\ 0 & \text{de lo contrario,} \end{cases}$$

$$a_{i,j} = \begin{cases} 1 & \text{si i es parte de la ruta j,} \\ 0 & \text{de lo contrario,} \end{cases}$$

y

$$c_j = \text{costo de usar la ruta j.}$$

Con estas notaciones, veamos el problema del agente viajero. Consideremos que el viajero debe recorrer n ciudades, las cuales pueden ser enumeradas como $0, 1, \ldots, n-1$. El principal objetivo es viajar desde la ciudad 0, ciudad de partida, visitar cada ciudad restante una y solo una vez y regresar a la ciudad 0. Luego, supondremos que se conoce la "distancia" entre cada par de ciudades, denotada como $c_{i,j}$, esta "distancia" también puede entenderse como "tiempo" o "costo". Y el viajero quiere visitar todas las ciudades minimizando la distancia total. La figura 1.1, nos muestra un ejemplo de siete ciudades, las cuales han sido enumeradas en el orden que fueron

visitadas. Si denotamos s_i como la i-ésima ciudad visitada, entonces el recorrido se describiría como $s_0 = 0, s_1, s_2, \ldots, s_{n-1}$.

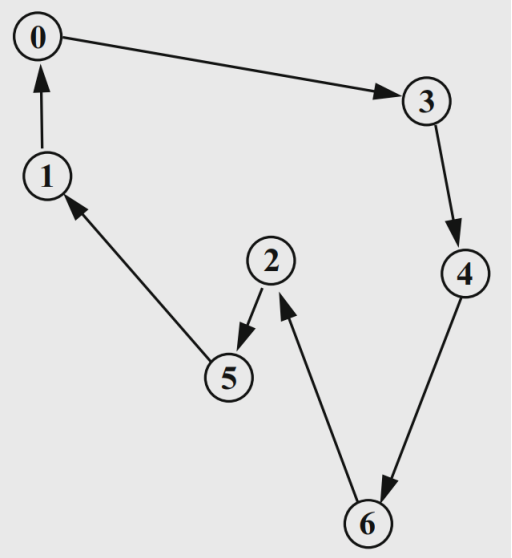


Figura 1.1. Un recorrido factible de siete ciudades. Sacado de (Vanderbei, 2014).

Sin embargo, este no es el único recorrido posible ya que el número total dependerá del número de formas que se pueda permutar las $n-1$ ciudades, es decir, $(n-1)!$. Y aún para ejemplos pequeños, como el de la figura (1.1), es (6)!, se tendrán 720 recorridos posibles. Por lo que, enumerar cada una de las soluciones posibles está fuera de discusión. Es por ello que nuestro objetivo es formular este problema como un programa entero, que nos permita resolverlo de forma más rápida que mediante la enumeración.

Recordando la notación, diremos que para cada (i, j) una variable de decisión x_{ij}, será 1 si luego de visitar la ciudad i, visita inmediatamente la ciudad j, 0 si no. Por lo que es fácil definir la función objetivo:

$$\text{Minimizar} \sum_i \sum_j c_{ij} x_{ij}.$$

Las cosas se complican un poco cuando debemos formular las restricciones, aunque algunas sean un poco obvias. Por ejemplo, luego de que se visita la ciudad i, se debe visitar una y solo una ciudad siguiente, también conocida como restricción de acceso, es decir,

$$\sum_j x_{ij} = 1, \qquad i = 0, 1, \ldots, n-1.$$

De forma similar, cuando el viajero llega a una ciudad, debe haber venido de una y solo una ciudad anterior, también conocida como restricción de procedencia, es

decir,

$$\sum_i x_{ij} = 1, \qquad j = 0, 1, \ldots, n-1.$$

El problema del agente viajero sería bastante fácil de resolver si las restricciones de acceso y de procedencia son suficientes para asegurar que las variables de decisión representan un recorrido. Pero este no es el caso, ya que estas restricciones no son suficientes, debido a que existe la posibilidad de que se formen subrecorridos disjuntos, como se puede apreciar en la figura (1.2).

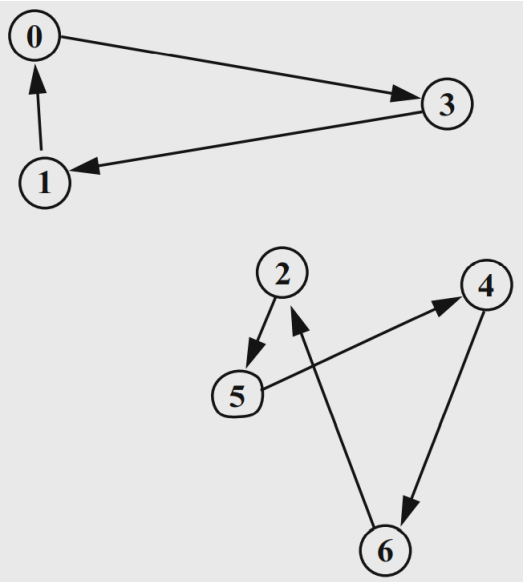


Figura 1.2. Un recorrido factible de 7 ciudades con subrecorridos disjuntos. Sacado de (Vanderbei, 2014).

Para que esto no suceda necesitaremos introducir más restricciones. Para ello, consideremos un recorrido específico

$$s_0 = 0, s_1, s_2, \ldots, s_{n-1}.$$

Sea t_i, para $i = 0, 1, \ldots, n$, el número de veces en que la ciudad i es visitada a lo largo del recorrido. De esta manera, vemos que $t_0 = 0$, $t_{s_1} = 1$, $t_{s_2} = 2$, etc. De forma general se tiene que

$$t_{s_i} = i, \qquad i = 0, 1, \ldots, n-1,$$

para que podamos pensar en los t_j's siendo el inverso de los s_i's. Para un recorrido de buena fe,

$$t_j = t_i + 1, \qquad \text{si} \quad x_{ij} = 1.$$

Además, cada t_i es un entero entre 0 y $n-1$. Por ello, se tienen las siguientes

restricciones para t_j

$$t_j \geq \begin{cases} t_i + 1 - n & \text{si} \quad x_{ij} = 0, \\ t_i + 1 & \text{si} \quad x_{ij} = 1. \end{cases}$$

lo cual se podría reescribir de la siguiente manera

$$t_j \geq t_i + 1 - n(1 - x_{ij}), \quad i \geq 0, j \geq 1, i \neq j.$$

Por otra parte, recordemos que estas restricciones se definieron para descartar la formación de subrecorridos, lo cual puede ser probado por contradicción. En consecuencia, podemos definir el problema del agente viajero como un problema de programación entera de la siguiente manera:

LM 1.4.1 Formulación TSP

Minimizar
$$\sum_{i,j} c_{ij} x_{ij} \tag{1.1}$$

sujeto a :

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 0, 1, \ldots, n-1, \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 0, 1, \ldots, n-1, \tag{1.3}$$

$$t_j \geq t_i + 1 - n(1 - x_{ij}), \qquad i \geq 0, j \geq 1, i \neq j, \tag{1.4}$$

$$t_0 = 0, \tag{1.5}$$

$$x_{ij} \in \{0, 1\}, \tag{1.6}$$

$$t_i \in \{0, 1, 2, \ldots\}. \tag{1.7}$$

Hay que tener en cuenta que, para n ciudades, se tendrán $n(n+1)$ variables en esta formulación. (Vanderbei, 2014)

1.4.6. Métodos que se han usado para resolver el TSP

Debido a que la solución a este problema resulta de gran complejidad se han desarrollado distintos métodos que buscan encontrar la ruta más eficiente que permita reducir costos o tiempo de traslado. Dichos métodos utilizan algoritmos aproximados que pueden ser del tipo: heurísticos o metaheurísticos.

Heurística se refiere a técnicas basadas en la experiencia para resolver problemas. Da una solución satisfactoria en un tiempo computacional razonable, el cual puede no ser óptimo. Las heurísticas específicas dependen de los problemas y son diseñadas para resolver un problema en particular. Ejemplos de este método incluyen juicios

intuitivos, estimación fundamentada o incluso sentido común. Muchos algoritmos, exactos o aproximados son heurísticas. (Du, 2010)

Metahuerística se refiere a una estrategia donde se aplican una o más heurísticas, permitiendo encontrar una solución de forma general, es decir, encuentra una solución factible a diferentes situaciones, ya que abarca mayor número de problemas.

También es importante mencionar que la distancia entre un nodo a otro puede ser simétrica, es decir, que la distancia desde N_1 hasta N_2 sea igual a la distancia desde N_2 hasta N_1, caso contrario las distancias no serían iguales, y es este el caso el que más se encuentra en la práctica.

A continuación, haremos una introducción a algunos métodos que se han usado para resolver el Problema del Agente Viajero.

Método del vecino más cercano

Se considera a este método un algoritmo heurístico, de hecho fue uno de los primeros en ser desarrrollado por K. Menger en el año 1930, como lo menciona Santamaría Carlos (2015), el cual no asevera encontrar la solución más óptima, pero si algunas buenas y en tiempos eficientes. Es en su simplicidad donde se encuentra su ventaja, dado que su estructura radica en la elección de un nodo cualquiera de la ruta, luego se debe hallar el más cercano al último nodo que fue añadido y también agregarlo a la ruta, lo último se debe repetir hasta haber visitado todos los nodos y finalmente regresar al nodo de partida.

En un estudio comparativo de métodos de solución al problema del agente viajero para el diseño de rutas de distribución en Empresa COMIX, desarrollada por Santamaría Carlos (2015), se puede ver que el método del vecino más cercano no es tan eficiente como se esperaría, en comparación de otros métodos que ahí se detallan. Para leer más sobre este ejemplo puede leer (Santamaría Carlos, 2015).

Heurístico	**Desviación del Óptimo** (%)	**Tiempo de Ejecución** (s)
Vecino más Próximo	18.6	0.3
Inserción más Lejana	9.9	35.4
Christofides	19.5	0.7
Ahorros	9.6	5.07

Tabla 1.2. Comparación de métodos: Diseño de rutas de distribución

A continuación, presentaremos algunas metaheurísticas aplicadas al problema del agente viajero:

Búsqueda tabú

Este método, propuesto por Fred Glover (1986), estudió las distintas soluciones que se encuentran fuera del espacio solución del óptimo local. Esto permite que se pueda cambiar de una posible solución, sin embargo, no asegura que sea mejor a la anterior, pero si permite escapar de óptimos locales y continuar en la búsqueda de la mejor solución. Para entenderlo mejor, este método itera cada movimiento y una vez realizado lo coloca dentro de una "lista tabú", esta hace que el siguiente movimiento que se vaya a realizar no se repita, dado que prohibe hacer el mismo movimiento que ya se encuentra en la lista. Además de escapar del óptimo local, tampoco entra en un ciclo repetitivo. Su memoria puede ser de corto o largo plazo. Cuando hablamos de corto plazo nos referimos a los movimientos que se han realizado recientemente, mientras que en la de largo plazo se encuentran los movimientos de mayor frecuencia de determinados eventos. Esta última es de gran ventaja cuando se requiere revisar regiones que no hayan sido exploradas previamente, véase (Hincapié Isaza et al., 2005).

Recocido Simulado

El método Recocido Simulado (S.A. por sus siglas en inglés), nos introduce al proceso de enfriamiento de metales y es que el término recocido hace relación al momento en que los metales en estado líquido se enfrían con el fin de garantizar un estado de energía mínima. Lo cual nos lleva a la relación de que el recocido simulado enfría la solución hasta obtener la más baja posible, entendiendo que si tenemos una solución y en su conjunto solución hay otra mejor, la va a tomar. Esto nos dice que gracias al cambio de solución se escapa del mínimo local, lo cual supera una de las desventajas que existe en la búsqueda local, véase (Minetti, 2003).

Por otro lado, se presentan los métodos exactos, los cuales obtienen la solución óptima. En este capítulo introduciremos el método Branch and Bound y Branch and Cut, mientras que en el siguiente capítulo explicaremos el método Cutting Plane. La importancia de estos métodos es que son usados en el algoritmo Concorde, el cual es nuestro objeto de estudio.

Branch and Bound

Este algoritmo se basa en dos principios, busca la solución óptima por derivación, es decir, partiendo el conjunto en subconjuntos (branch), e intenta podar la enumeración delimitando el valor objetivo de los subconjuntos o subproblemas que fueron generados en la partición (bound). En otras palabras, este método parte de una bifurcación, la cual crea dos subproblemas al restringir el rango de una variable, dicha estrategia se la conoce como ”ramificación variable". Para acotar el valor objetivo de un subproblema se resuelve su relajación de programación lineal natural, a esta estrategia se la conoce como ”acotación de programación lineal". Estas estrategias son ampliamente usadas en los solucionadores de programación de enteros, véase

(Conforti M., 2014).

Este algoritmo mantiene una lista de problemas de programación lineal obtenidos al relajar los requisitos de integridad en las variables $x_j, j = 1, ..., n$, e imponer restricciones lineales, por ejemplo, límites en las variables $x_j \leq l_j$ o $x_j \geq S_j$. Cada uno de estos programas lineales corresponde a un nodo del árbol de enumeración. Para un nodo n_i, se denota a z_i el valor del programa lineal correspondiente LP_i. Se denota $\mathcal{L}$ a la lista de nodos que aún quedan por resolver, es decir, que aún no han sido ramificados, ni acotados. Sea z_* un límite inferior del valor óptimo z^*. Con estos datos, presentaremos a continuación algoritmo de Branch and bound.

Algorithm 1 Algoritmo de Branch and Bound

Paso 0. Definición de variables
$\mathcal{L} := \{n_0\}$, $z_* := -\infty$, $(x^*, y^*) := 0$.

Paso 1. ¿Terminar?
Si $\mathcal{L} = 0$, la solución (x^*, y^*) es óptima.

Paso 2. Seleccionar nodo
Elija un nodo n_i en $\mathcal{L}$ y eliminelo de $\mathcal{L}$.

Paso 3. Acotar
Resuelva LP_i. Si es inviable, vaya al Paso 1. De lo contrario, tome (x^i, x^j) como la solución óptima de LP_i y z_i su valor objetivo.

Paso 4. Cortar
Si $z_i \leq z_*$, vaya al Paso 1.
Si (x^i, x^j) es factible, denote $z_* := z_i$, $(x^*, y^*) := (x^i, x^j)$ y vaya al Paso 1.
De lo contrario.

Paso 5. Ramificar
De LP_i, construir $k \geq 2$ programas lineales $LP_{i_1}, ..., LP_{i_k}$, con regiones factibles más pequeñas cuya unión no contiene a (x^i, x^j), pero si a todas las soluciones de LP_i, con $z \in \mathbb{Z}^n$. Agregue los nuevos nodos correspondientes $n_{i_1}, ..., n_{i_k}$ a $\mathcal{L}$ y vaya al Paso 1.

Se puede observar que varias opciones se dejan abiertas, como el criterio de selección de nodos en el paso 2 y la estrategia de ramificación en el paso 5. Por ello, se tienen cuatro problemas que requieren atención cuando se resuelven programas enteros mediante algoritmos de ramificación y acotación, como lo son:

- Formulación, para que la brecha $z_0 - z^*$ sea pequeña.
- Heurística, para encontrar un límite inferior z_* que sea bueno.
- Derivación.
- Selección de nodos.

Branch and Cut

En este método es crucial la estrechez del límite superior al momento de podar el árbol de enumeración (Paso 4). Por lo que se podría calcular los límites superiores más estrictos aplicando el enfoque de plano de corte a los subproblemas, este enfoque se desarrollará en la **sección 2.3**. Esto nos lleva a desarrollar el enfoque del método de ramificación y corte, que es considerado en la actualidad como el más exitoso al momento de resolver programas enteros. Este se obtiene agregando un paso de plano de corte antes del paso de ramificación en el algoritmo de ramificación y acotación del método de Branch and Bound descrito anteriormente, véase (Conforti M., 2014). El algoritmo de Branch and Cut estará dado por

Algorithm 2 Algoritmo de Branch and Cut

Paso 0. Definición de variables
$\mathcal{L} := \{n_0\}$, $z_* := -\infty$, $(x^*, y^*) := 0$.
Paso 1. ¿Terminar?
Si $\mathcal{L} = 0$, la solución (x^*, y^*) es óptima.
Paso 2. Seleccionar nodo
Elija un nodo n_i en $\mathcal{L}$ y eliminelo de $\mathcal{L}$.
Paso 3. Acotar
Resuelva LP_i. Si es inviable, vaya al Paso 1. De lo contrario, tome (x^i, x^j) como la solución óptima de LP_i y z_i su valor objetivo.
Paso 4. Cortar
Si $z_i \leq z_*$, vaya al Paso 1.
Si (x^i, x^j) es factible, denote $z_* := z_i$, $(x^*, y^*) := (x^i, x^j)$ y vaya al Paso 1.
De lo contrario.
Paso 5. ¿Agregar cortes?
Decidir fortalecer la formulación LP_i o para remificar.
En el primer caso, fortalecer LP_i agregando planos de corte y regrese al Paso 3.
En el segundo caso, vaya al Paso 6.
Paso 6. Ramificar
De LP_i, construir $k \geq 2$ programas lineales $LP_{i_1}, ..., LP_{i_k}$, con regiones factibles más pequeñas cuya unión no contiene a (x^i, x^j), pero si a todas las soluciones de LP_i, con $z \in \mathbb{Z}^n$. Agregue los nuevos nodos correspondientes $n_{i_1}, ..., n_{i_k}$ a $\mathcal{L}$ y vaya al Paso 1.

Se debe analizar si se desea añadir cortes en el nuevo Paso 5, esto dependerá del éxito de los cortes que fueron agregados previamentee y de las características de los nuevos cortes. Es más común encontrar varias rondas de corte en el nodo n_0, ya que más abajo en el árbol de enumeración se generará menos o ningún corte.

Para un estudio más profundo de estos métodos véase (Conforti M., 2014).

Capítulo 2

Metodología

2.1. Datos de la investigación

La presente investigación es de tipo cuantitativa, dado que se ha trabajado con datos que permiten generar una comparación de métodos en relación al algoritmo Concorde, los mismos que han permitido verificar efectividad y rapidez de las soluciones. Se menciona además que los datos 2, 3 y 4 fueron extraídos de la página del TSP de la universidad de Waterloo, los cuales están disponibles para estudios académicos, véase (Cook, 2009). La razón de trabajar con estos datos se debió a la complejidad en el levantamiento de información.

2.1.1. Datos 1: 23 instancias / Caso real: Provincias del Ecuador

En esta primera sección se escogieron las coordenadas que coorresponden a cada provincia del Ecuador, tal que bajo cualquier contexto se pueda hallar la ruta más corta entre ellas. Se consideraron 23 de las 24 provincias del país sin incluir Galápagos como se muestra a continuación:

Nodo	Coord. x	Coord. y
1	753.582001	596.483238
2	789.873491	804.724857
3	798.806781	875.716318
4	782.615193	511.293485
5	792.106814	624.879822
6	814.440038	710.069575
7	767.540267	570.453035
8	767.540267	355.112270
9	846.823214	544.422833
10	811.648385	918.311194
11	716.173850	482.896900
12	809.415063	421.370967

Nodo	Coord. x	Coord. y
13	777.590218	686.405755
14	741.298728	504.194339
15	861.898140	814.190385
16	774.798565	426.103731
17	764.190283	771.595508
18	871.948092	686.405755
19	766.423605	927.776723
20	741.857058	279.388045
21	732.365438	698.237665
22	777.590218	180.000000
23	747.440364	184.732764

Tabla 2.1. Valores de 23 instancias

2.1.2. Datos 2: 76 instancias / Propuesto por Padberg y Rinaldi

Nodo	Coord. x	Coord. y
1	3600	2300
2	3100	3300
3	4700	5750
4	5400	5750
5	5608	7103
6	4493	7102
7	3600	6950
8	3100	7250
9	4700	8450
10	5400	8450
11	5610	10053
12	4492	10052
13	3600	10800
14	3100	10950
15	4700	11650
16	5400	11650
17	6650	10800
18	7300	10950
19	7300	7250
20	6650	6950
21	7300	3300
22	6650	2300
23	5400	1600
24	8350	2300
25	7850	3300
26	9450	5750
27	10150	5750
28	10358	7103
29	9243	7102
30	8350	6950
31	7850	7250
32	9450	8450
33	10150	8450
34	10360	10053
35	9242	10052
36	8350	10800
37	7850	10950
38	9450	11650

Nodo	Coord. x	Coord. y
39	10150	11650
40	11400	10800
41	12050	10950
42	12050	7250
43	11400	6950
44	12050	3300
45	11400	2300
46	10150	1600
47	13100	2300
48	12600	3300
49	14200	5750
50	14900	5750
51	15108	7103
52	13993	7102
53	13100	6950
54	12600	7250
55	14200	8450
56	14900	8450
57	15110	10053
58	13992	10052
59	13100	10800
60	12600	10950
61	14200	11650
62	14900	11650
63	16150	10800
64	16800	10950
65	16800	7250
66	16150	6950
67	16800	3300
68	16150	2300
69	14900	1600
70	19800	800
71	19800	10000
72	19800	11900
73	19800	12200
74	200	12200
75	200	1100
76	200	800

Tabla 2.2. Valores de 76 instancias

En la tabla anterior se puede visualizar que se ha considerado el caso de 76 nodos o ciudades, propuesto por Padberg y Rinaldi. Problema que causaba dificultad en los investigadores de la época, dado que los algoritmos propuestos inicialmente tomaban demasiado tiempo de ejecución y en otros casos no eran ejecutables.

2.1.3. Datos 3: 318 instancias / Propuesto por Crowder y Padberg

Similarmente, se verificaron los métodos, a comparar en el presente proyecto, usando 318 nodos. Problema que fue propuesto por Crowder y Padberg cuando tenían el objetivo de hallar una solución óptima que permita recorrer puntos de una tarjeta de circuito impreso. Dado que la cantidad de datos ha sido considerablemente grande, se omite anexar cada uno de ellos en el presente trabajo.

2.1.4. Datos 4: 1002 instancias / Propuesto por Padberg y Rinaldi

Finalmente, se realizó la respectiva comparación utilizando los datos propuestos por Padberg y Rinaldi para recorrer 1002 ciudades. El esquema general del problema se relaciona con el análisis desarrollado bajo el supuesto de hallar el recorrido más óptimo entre dicha cantidad de ciudades. La mayor dificultad en este caso fue ejecutar cualquiera de los métodos con la cantidad de nodos mencionada.

2.2. Software donde se desarrolla

Como se ha mencionado previamente, el algoritmo de Concorde ha sido desarrollado inicialmente en lenguaje C, el mismo que ha tenido varias mejoras hasta los últimos tiempos. Cabe considerar que dada la cantidad de funciones que maneja el software, existe una gran dificultad en la implementación directa mediante el código propuesto por el autor. En otros casos, la falta de conocimiento del detalle de su funcionamiento limita el uso de este algoritmo.

Actualmente, existe un programa ejecutable, denominado "Graphical User Interface", el mismo que ha permitido ejecutar datos relacionados al problema de TSP con solución directa a través del algoritmo Concorde que viene integrado en la interfaz gráfica. Cualquier instancia, puede ser analizada mediante el algoritmo de Concorde usando este graficador, donde además se obtiene la solución óptima. (Cook, 2015)

Similarmente, para las instancias que se requirieron para registrar la solución del problema usando los algoritmos adicionales al Concorde, se utilizó el software Mathematica. El mismo que provee un entorno de computación técnica para sus usuarios. En este caso, se lo ha utilizado para calcular la matriz distancia que permite aplicar los algoritmos estudiados para la respectiva comparación. (Wolfram, 2023)

Finalmente, se implementó el lenguaje de programación Python, que es accesible de manera gratuita y se puede manipular con pocos conocimientos de programación, además, posee una amplia variación de aplicaciones acorde al área de estudio. Dicho lenguaje se ha utilizado para implementar el código del método Cplex en instancias pequeñas, es decir, cantidades pequeñas de nodos. El mismo que permitió comparar los resultados del algoritmo Concorde.

2.3. Funcionamiento del algoritmo Concorde

Para entender el funcionamiento del algoritmo Concorde se requiere de una breve explicación sobre el método de plano de corte, la cual se detalla a continuación:

2.3.1. Cutting Plane

Considere el programa entero

$$MILP: \quad max\{cx + hy : (x, y) \in S\}$$

donde, $S := \{(x.y) \in \mathbb{Z}^n_+ \times \mathbb{R}^n_+ : Ax + Gy \leq b\}$. Sea P_0 la relajación lineal natural de S. Se plantea resolver el programa lineal

$$max\{cx + hy : (x, y) \in P_0\}. \tag{2.1}$$

Sea z_0 el valor óptimo y (x^0, y^0) una solución óptima, también se puede suponer que es una básica solución óptima (la noción de esto puede verse en el capítulo 3 en (Conforti M., 2014)). Dichas soluciones pueden ser calculadas usando algoritmos de programación lineal estándar.

Ahora, considere el caso cuando la solución (x^0, y^0) no está en S. En este caso se trata de encontrar una desigualdad $\alpha x + \gamma y \leq \beta$, que se satisface en todos los puntos de S, tal que $\alpha x^0 + \gamma y^0 > \beta$, cuya existencia está garantizada cuando (x^0, y^0) es una solución básica de (2.1). Si esta desigualdad válida es violentada por (x^0, y^0), entonces se convierte en un plano de corte separando (x^0, y^0) de S. Sea $\alpha x + \gamma y \leq \beta$ un plano de corte, se define

$$P_1 = P_0 \cap \{(x, y) : \alpha x + \gamma y \leq \beta\}.$$

Resolver el programa lineal

$$max\{cx + hy : (x, y) \in P_1\} \tag{2.2}$$

es tan bueno como un límite superior en el valor z^* como z_0, mientras que $(x^0, y^0) \notin P_1$. La aplicación recursiva de esta idea conduce al plano de corte aproximado.

Algorithm 3 Algoritmo Cutting Plane

i=0, repetir:
Paso recursivo: Resolver (2.2)
Si la solución básica óptima asociada $(x^i, y^i) \in S$, deténgase.
Caso contrario, resuelva el problema de separación.
Encuentre un plano de corte $\alpha x + \gamma y \leq \beta$ separando (x^i, y^i) de S.
Sea $P_{i+1} := P_i \cap \{(x.y) : \alpha x + \gamma y \leq \beta\}$ y repita el paso recursivo.

Algorithm 4 Algoritmo Cutting Plane

i=0, repetir:
Paso recursivo: Resolver el programa lineal, $max\{cx + hy : (x, y) \in P_1\}$
Si la solución básica óptima asociada $(x^i, y^i) \in S$, deténgase.
Caso contrario, resuelva el problema de separación.
Encuentre un plano de corte $\alpha x + \gamma y \leq \beta$ separando (x^i, y^i) de S.
Sea $P_{i+1} := P_i \cap \{(x.y) : \alpha x + \gamma y \leq \beta\}$ y repita el paso recursivo.

El tema central en la programación entera son los problemas de separación que se resuelven con el algoritmo de plano de corte, de aquí la importancia del estudio para comprender el funcionamiento del algoritmo Concorde, descrito en la siguiente subsección.

2.3.2. Esquema de funcionamiento del algoritmo

A continuación, de forma general, se describirá el análisis realizado sobre el algoritmo Concorde, la mayor parte de información se ha obtenido de (Applegate et al., 2006).

Como previamente se ha mencionado el algoritmo de Concorde se basa en el método de ramificación y corte **(sección 1.4.5)** . Donde, además, se aplica el algoritmo conocido como plano de corte dentro de la búsqueda de ramas que requiere el algoritmo principal para reducir en subproblemas el objetivo inicial.

De forma general, al recorrer el paso de bifurcación se elige un subconjunto apropiado $S \subseteq V$, donde V es el conjunto de vértices o nodos del problema y S es el subcojunto de nodos obtenidos acorde a la bifurcación. A partir de ello, dos subproblemas se crean al considerar las siguientes condiciones:

$$x(\delta(S)) = 2 \quad \& \quad x(\delta(S)) \geq 4,$$

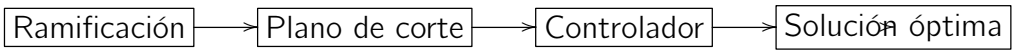
donde $\delta(S)$ es el conjunto de aristas con terminación en S.

El paso descrito, permite crear un árbol de búsqueda, donde el nodo raíz será el nodo inicial. En este paso, se aplica además el algoritmo de plano de corte.

Se puede pensar en la forma por la cual el algoritmo es capaz de analizar una gran cantidad de nodos, lo mismo que para los conocedores del área computacional les llevaría a considerar un proceso similar al almacenamiento de memoria a través del sistema. A continuación, se detalla de forma general el mecanismo computacional que se aplica para cualquier solucionador de problemas TSP, basados en la metodología del algoritmo Concorde.

Dado que Concorde se desarrolla mediante el método de ramificación y corte, el diseño de controlador basado en dicho método, busca generar un mecanismo que permita el flujo de los bordes de corte útiles para la solución, generando planos o columnas de corte según lo que corresponda, con la intención de evitar nodos que no optimicen la solución. Para desarrollar este mecanismo, Concorde usa dos bucles en su desarrollo. El bucle interno permite realizar corte en los bordes más cercanos según su costo, y el bucle externo que considera el costo total del borde previamente cortado. Para dicho proceso, Concorde utiliza una función de evaluación simple que permita recorrer el problema hasta la mejor solución posible.

Se presenta a continuación un pequeño diagrama que permite evidenciar las etapas que involucra el algoritmo:



2.4. Criterios de comparación con otros métodos

Para la investigación desarrollada, se consideraron algunos criterios que permitían verificar la veracidad que se tiene al suponer al algoritmo Concorde uno de los más eficaces que se han desarrollado dentro del área.
Por ello, para el desarrollo del análisis, se escogieron los siguientes criterios:

1. El tiempo de ejecución del programa por cada método, que ha sido medido mediante un cronómetro que consideró el momento de inicio y fin del desarrollo de cada algoritmo.

2. La calidad de la solución en relación a la ruta más corta, que ha sido evaluada comparando el resultado de ruta por cada problema ingresado.

3. Rango de dificultad de las características computacionales de cada método.

A continuación se puede visualizar una breve descripción de los métodos que se compararon con el algoritmo Concorde.

2.4.1. Método LinKer

En la búsqueda de los algoritmos más exitosos hasta la fecha se encontró al simple y elegante algoritmo de Lin y Kernighan. El cual comenzó su estudio con limitación de 110 ciudades (instancia muy pequeña para la actualidad) y en las últimas décadas se han realizado amplias aplicaciones para el TSP.

Se considera a este método como uno de los más efectivos para encontrar la solución óptima. Esta heurística comienza con un tour T factible, luego examina un vecindario de tours $T_1, T_2, ..., T_n$, aplicando cambios de k arcos de T. Estos cambios se aplican para disminuir la longitud del tour y el algoritmo acaba cuando ya no encuentra posibles mejoras. Se dice que el tour es k-opt, Lin realizó un estudio para los casos $k = 2$ y $k = 3$, en los que encontró buenas soluciones para el TSP en tiempos de cómputo muy cortos.

Lin propuso y probó una mejora del método de inversión de Croes. Para describir esto, se parte de la idea de Flood, el vecino más cercano, se elige una ciudad de partida y se visita la ciudad más cercana que aún no haya sido visitada. En el que, para cualquier recorrido óptimo, $(i_0, ..., i_{n-1})$, con n ciudades, y para cualquier $0 \leq p < q < n$, se tiene

$$c_{i_{p-1}} c_{i_p} + c_{i_q} c_{i_{q+1}} \leq c_{i_{p-1}} c_{i_q} + c_{i_p} c_{i_{q+1}}. \tag{2.3}$$

Un par (p, q) que violente (2.3) es llamada "par que se intersecta", y su método consistía en fijar cualquiera de esos pares, como se ilustra en la figura 2.1, hasta que el recorrido no tuviera intersecciones.

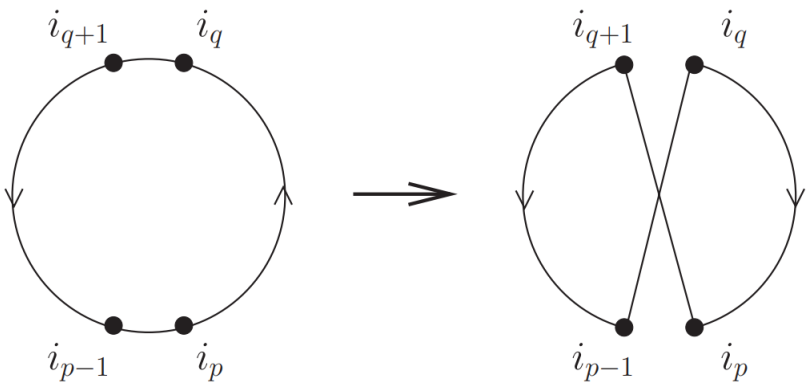


Figura 2.1. Arreglando un par que se cruza. Sacado de (Applegate et al., 2006).

Croes usó esta idea para estudiar un algoritmo de búsqueda exhaustivo para el TSP. En el cual se describe un procedimiento para encontrar un recorrido sin intersección mediante una secuencia de "inversiones". Su método consistía en que si (p, q) se cruzan en el recorrido

$$(i_0, ..., i_{p-1}, i_p, ..., i_q, i_{q+1}, ..., i_{n-1}),$$

entonces el par se puede arreglar invirtiendo la subsecuencia $(i_p, ..., i_q)$, cuyo recorrido

sería

$$(i_0, ..., i_{p-1}, i_q, i_{q-1}, ..., i_{p+1}, i_p, i_{q+1}, ..., i_{n-1}).$$

A esta operación se le llama $flip(p, q)$; se asume que los recorridos están orientados, así $flip(p, q)$ está bien definida.

A partir de esto, Lin propuso no solo voltear la subsecuencia, sino también reinsertarla (como estaba o invertida) entre dos ciudades adyacentes en el recorrido, siempre y cuando tal movimiento resultase en un recorrido de menor costo. Si bien, la complejidad del algoritmo aumenta, Lin probó que esto produce mejores recorridos.

Lin describe recorridos sin intersecciones y recorridos que son óptimos con respecto a los giros y la inserción. Denominó a un giro $k-opt$ si no es posible obtener un recorrido de menor costo reemplazando cualquier k de sus aristas (considerando el recorrido como un ciclo de un grafo) por cualquier otro conjunto de k aristas. Por lo tanto, un recorrido no tiene intersección si y solo si es $2-opt$. Además, no es difícil ver que un recorrido es óptimo con respecto a los giros e inserciones si y solo si es $3-opt$. El algoritmo de Croes y el algoritmo de Lin se conocen comúnmente como $2-opt$ y $3-opt$, respectivamente.

Lo próximo sería intentar $k-opt$ para $k > 3$. Lin descubrió que, a pesar de que el tiempo de cómputo era mucho mejor, los recorridos obtenidos no eran notablemente mejores que los producidos por $3-opt$. Como alternativa, (Du, 2010) desarrollaron un algoritmo que a veces se denomina $k-opt\ variable$. Este algoritmo se basa en un método de búsqueda efectiva que realiza una secuencia de volteretas, posiblemente bastante larga, de modo que cada subsecuencia pueda conducir a la búsqueda de un recorrido de menor costo. Si la búsqueda logra encontrar el recorrido mejorado, entonces se realiza la secuencia de volteretas y se inicia una nueva búsqueda. Si esto no sucede, se descartan las volteretas antes de iniciar la nueva búsqueda, teniendo cuidado de no repetir la misma secuencia fallida. El algoritmo termina cuando todos los puntos de partida de la búsqueda han resultado infructuosos. Un esquema de búsqueda del método de Lin y Kernighan es:

Algorithm 5 Esquema de búsqueda del método Lin-Ker

delta = 0
while existe vértice prometedor
do elige un vértice prometedor i_q;
$delta = delta + c(i_{p_1}, i_p) - c(i_p, i_{q+1}) + c(i_q, i_{q+1}) - c(i_q, i_{p_1})$;
add $flip(i_p, i_q)$ a la secuncia de flip;
end

2.4.2. Solver: CPLEX

El solver CPLEX que está basado en los solucionadores de programación matemática, ha sido utilizado para comparar los resultados de los datos previos en relación al

algoritmo Concorde y al método de LinKer. Su tecnología permite optimizar los resultados con base en las decisiones del problema que mejoren la eficiencia, la reducción de costos y la rentabilidad. Además, el algoritmo que se utiliza permite desarrollar problemas de programación del tipo: lineal, entera combinada, cuadrática o limitada cuadráticamente.

Capítulo 3

Resultados

Para la ejecución de todos los programas que se usaron en este proyecto se trabajó desde una computadora HP, procesador AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz, RAM instalada 16 GB, con edición Windows 11 Home.

3.1. Resultado Solver: CPLEX

Para realizar una comparación respecto al tiempo de ejecución entre un método exacto comercial y el Algoritmo Concorde, se ha usado un código en python que trabaja con el solver CPLEX. Dicho código brindó soluciones en las 2 primeras instancias propuestas, mientras que en las 2 restantes la computadora ejecutó el programa por más de 4 horas sin tener una solución, dichos resultados se presentan en la siguiente tabla:

Caso Propuesto	**Nodos**	**Solución Óptima**	**Tiempo de Ejecución** (s)
Provincias del Ecuador	23	1792	4
Padberg y Rinaldi	76	108159	5109
Crowder y Padberg	318	-	> 14400
Padberg y Rinaldi	1002	-	> 14400

Tabla 3.1. Solución óptima para las distintas instancias

Se obtuvo la siguiente gráfica para el caso real propuesto de los 23 nodos:

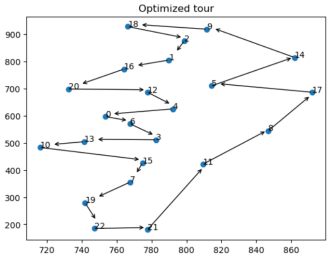
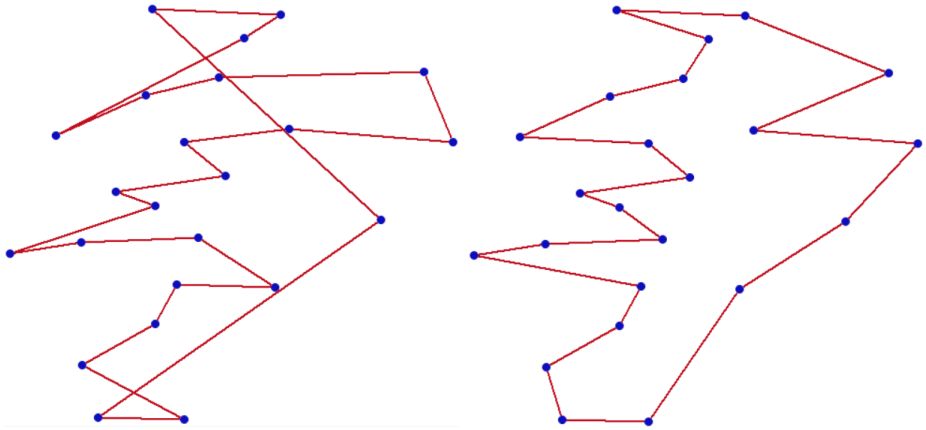


Figura 3.1. Solver CPLEX
Solución óptima: 1792

Los siguientes resultados fueron extraídos desde el Graphical User Interface, mismo que nos permitió hacer el análisis del método del vecino más cercano, método LinKer y el algoritmo Concorde, los cuales se presentan a continuación.

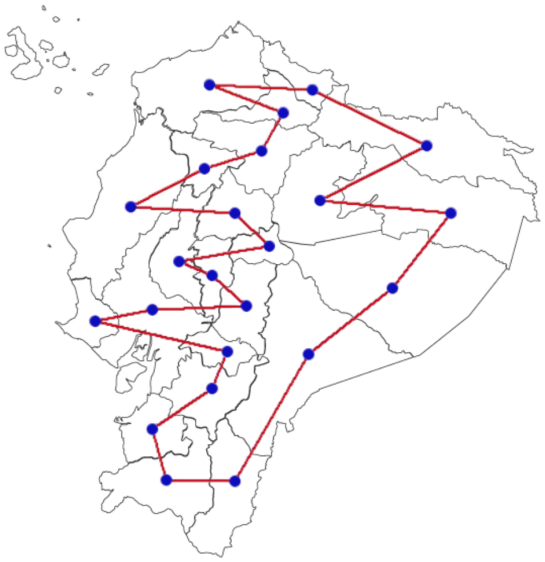
3.2. Resultado Datos 1: 23 instancias / Caso real

Se obtuvieron las coordenadas (latitud,longitud) para cada provincia de Ecuador, las cuales fueron convertidas a coordenadas (x,y) y posteriormente se ingresaron en el interfaz, obteniéndose las siguientes gráficas:



(a) Método Vecino más cercano
Solución: 2128

(b) Método LinKer
Solución: 1792



(c) Algoritmo Concorde
Solución: 1792

Figura 3.2. Soluciones para 23 instancias

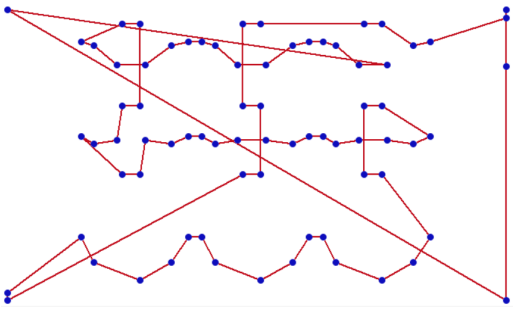
Luego, hicimos un análisis respecto a la desviación del óptimo (%) y el tiempo de

ejecución (s) de cada método, donde se puede apreciar que para las 23 instancias el método del vecino más cercano está desviado del óptimo por un $18,75\,\%$, mientras que por el método LinKer se obtuvo la solución óptima, como se puede ver en la siguiente tabla:

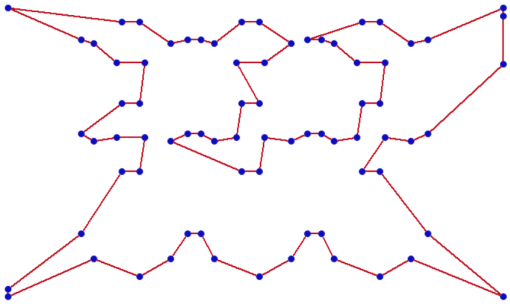
Método	**Desviación del Óptimo** (%)	**Tiempo de Ejecución** (s)
Vecino más cercano	18,75	1
Lin Kernighan	0	1
Concorde	0	2

Tabla 3.2. Comparación de métodos: Caso real - 23 Provincias del Ecuador

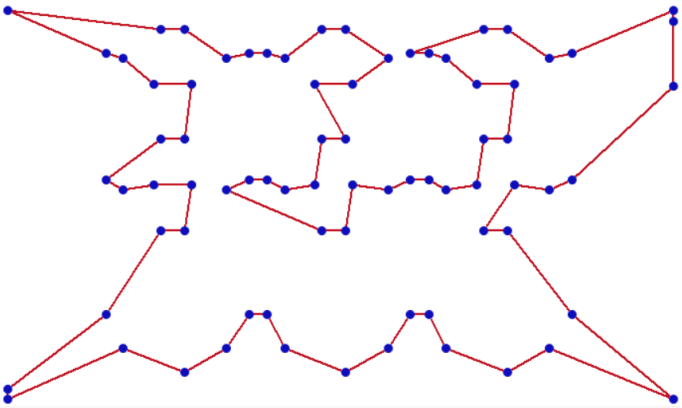
3.3. Resultado Datos 2: 76 instancias / Padberg y Rinaldi



(a) Método Vecino más cercano
Solución: 156883



(b) Método LinKer
Solución: 108159



(c) Algoritmo Concorde
Solución: 108159

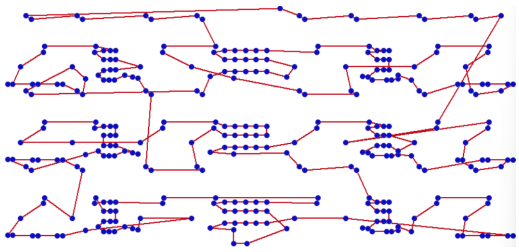
Figura 3.3. Soluciones para 76 instancias

Se puede apreciar que para las 76 instancias el método del vecino más cercano está desviado del óptimo por un $45,05\,\%$, mientras que por el método LinKer se obtuvo la solución óptima, como se puede ver en la siguiente tabla:

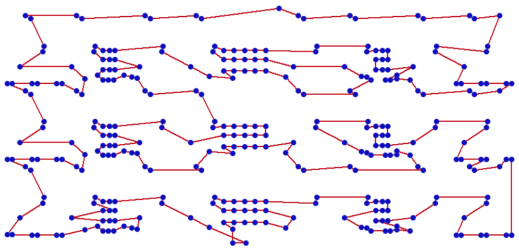
Método	**Desviación del Óptimo** (%)	**Tiempo de Ejecución** (s)
Vecino más cercano	45,05	1
Lin Kernighan	0	1
Concorde	0	3

Tabla 3.3. Comparación de métodos: 76 instancias - Propuesto por Padberg y Rinaldi

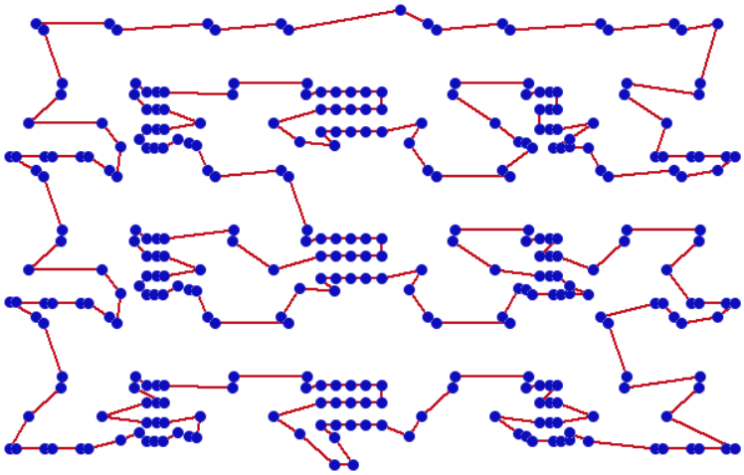
3.4. Resultado Datos 3: 318 instancias / Crowder y Padberg



(a) Método Vecino más cercano
Solución: 51528



(b) Método LinKer
Solución: 42946



(c) Algoritmo Concorde
Solución: 42029

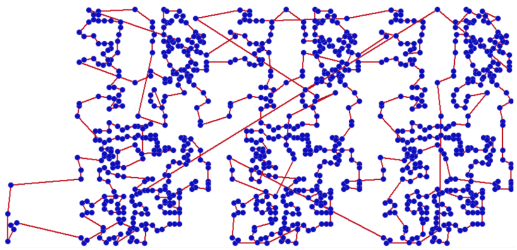
Figura 3.4. Soluciones para 318 instancias

Podemos notar que para las 318 instancias el método del vecino más cercano y el método LinKer están desviados del óptimo por un $21,65\,\%$ y $0,97\,\%$, respectivamente, como se puede ver en la siguiente tabla:

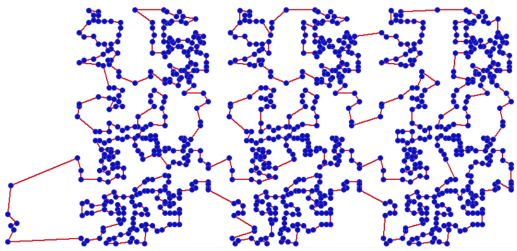
Método	**Desviación del Óptimo** (%)	**Tiempo de Ejecución** (s)
Vecino más cercano	21,65	1
Lin Kernighan	0,97	1
Concorde	0	6

Tabla 3.4. Comparación de métodos: 318 instancias - Propuesto por Crowder y Padberg

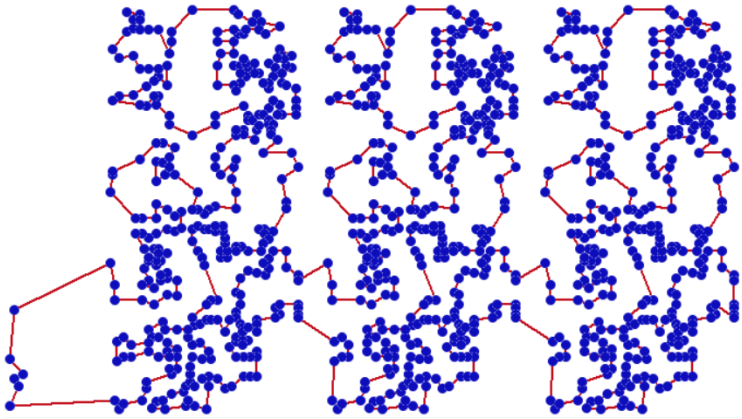
3.5. Resultado Datos 4: 1002 instancias / Padberg y Rinaldi



(a) Método Vecino más cercano
Solución: 338047



(b) Método LinKer
Solución: 261674



(c) Algoritmo Concorde
Solución: 259045

Figura 3.5. Soluciones para 1002 instancias

Finalmente, para las 318 instancias el método del vecino más cercano y el método LinKer están desviados del óptimo por un $30,50\,\%$ y $1,01\,\%$, respectivamente, como se puede ver en la siguiente tabla:

Método	**Desviación del Óptimo** (%)	**Tiempo de Ejecución** (s)
Vecino más cercano	30,50	2
Lin Kernighan	1,01	2
Concorde	0	22

Tabla 3.5. Comparación de métodos: 1002 instancias - Propuesto por Padberg y Rinaldi

En resumen podemos apreciar las soluciones de cada método para todas las instancias en el siguiente gráfico:

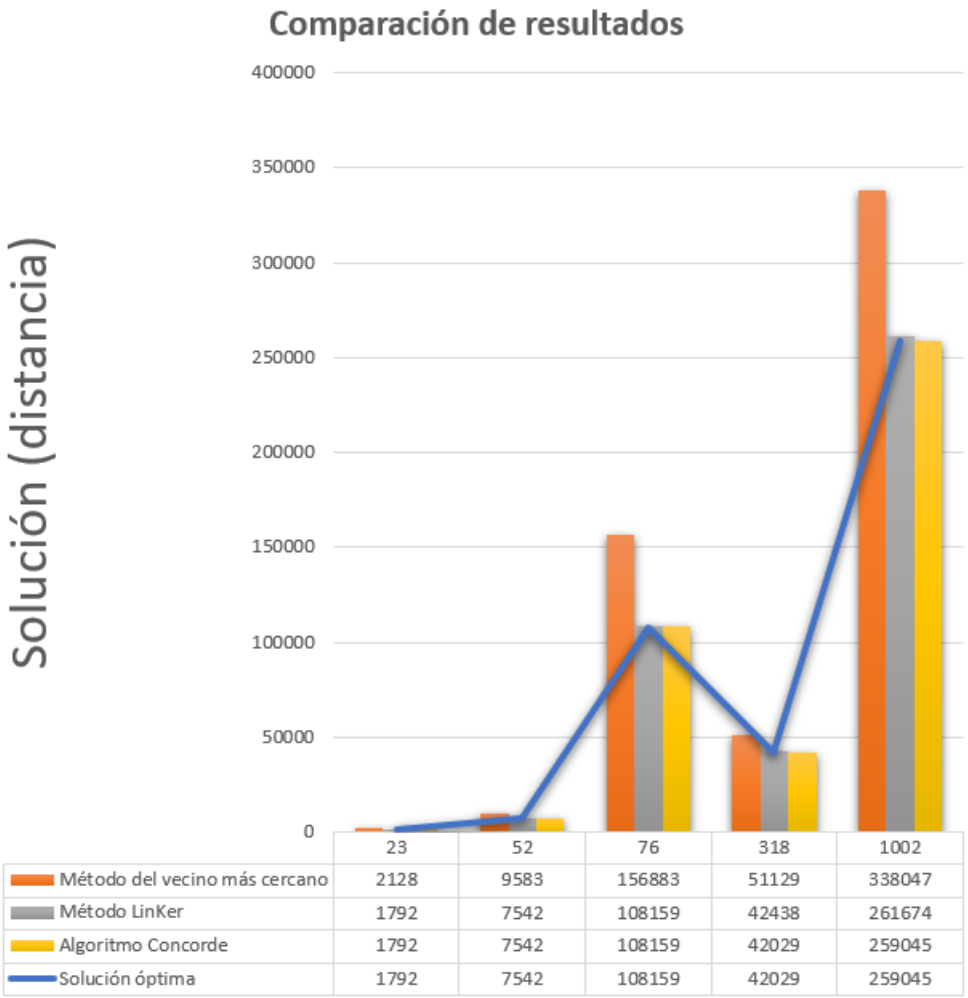


Figura 3.6. Resultados para todas las instancias

Capítulo 4

Conclusiones y recomendaciones

4.1. Conclusiones

Con el propósito de hacer un estudio cronológico del Problema del Agente Viajero Simétrico y de las fortalezas que presenta el Algoritmo Concorde en la solución de este problema, utilizando el solver CPLEX para comparar los métodos exactos y el Graphical User Interface para el análisis de los otros métodos, se concluye que:

- El método del vecino más cercano no es tan bueno como se esperaba ya que las desviaciones de los óptimos para cada instancia eran altas, además la solución no era única.
- El método LinKer logró llegar a la solución óptima en 2 de las 4 instancias propuestas en este trabajo, sin embargo en las 2 restantes las desviaciones eran menores al 2 %. Por lo cual, se considera un método bastante aproximado para instancias grandes.
- Por último, el algoritmo Concorde refleja lo rentable que sería en la implementación del Problema del Agente Viajero Simétrico, dado que los costos computacionales son bajos. Por ello se lo considera como el más eficaz para resolver instancias grandes en tiempos razonablemente cortos, mismo que fue comparado con el solver CPLEX, el cual tomó muchísimo tiempo más para resolver las instancias propuestas.

Similarmente, acorde a la literatura estudiada, se reconoce que el algoritmo Concorde se puede aplicar a cualquier situación que cumpla con las condiciones del problema, por lo que no se limita su uso únicamente a los tours o recorridos de ciudades, provincias o estados; sino que posee una amplia adaptabilidad a otros campos de estudio. Del mismo modo, permite la eficaz optimización de recursos como respuesta de hallar el recorrido óptimo del problema. A pesar de ello, puede resultar un reto para los investigadores comprender la literatura que involucra el entendimiento del funcionamiento del algoritmo, dado que el autor principal tuvo que incluir más de 700 funciones para su desarrollo.

4.2. Recomendaciones

Acorde a lo propuesto a lo largo de este trabajo y frente a las dificultades descubiertas en su desarrollo, se recomienda para futuras investigaciones o proyectos:

- Obtener una cuenta pagada o una cuenta académica del ILOG CPLEX Optimization Studio, si se desea trabajar con instancias muy grandes, que permita obtener las soluciones óptimas con menor dificultad.
- Trabajar con la matriz de distancia de las coordenadas redondeada (Wolfram Mathematica), puesto que el Algoritmo Concorde trabaja con distancias cerradas.
- Hacer una comparación frente a otros métodos exactos presentes en la literatura, para una mejor visualización de fortalezas del Algoritmo Concorde.
- Aplicar el algoritmo y su respectivo análisis a un caso real, distinto al presentado en este trabajo. Mismo que posea un propósito académico, cultural o social, de mayor impacto en la literatura, cuyos resultados muestren el gran alcance del algoritmo para la optimización de recursos.

Bibliografía

[1] Academic. https://es-academic.com/dic.nsf/eswiki/172753. Accessed: 2022-8-31. 2022.

[2] David L. Applegate et al. *The Traveling Salesman Problem A Computational Study*. Princeton, New Jersey 08540: Princeton university press, 41 William Street, 2006. ISBN: 978069112993.

[3] Zambelli G. Conforti M. Cornuéjols G. *Integer Programming*. London, UK: Springer, 2014. ISBN: 0072-5285.

[4] William J. Cook. https://www.math.uwaterloo.ca/tsp/data/index.html. Accessed: 2009-02-31. 2009.

[5] William J. Cook. https://www.math.uwaterloo.ca/tsp/concorde/index.html. Accessed: 2015-3-31. 2015.

[6] William J. Cook. https://www.math.uwaterloo.ca/tsp/index.html. Accessed: 2022-8-31. 2022.

[7] William J. Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton, New Jersey 08540: Princeton university press, 41 William Street, 2012. ISBN: 9870691152707.

[8] Robert E. Bixby David L. Applegate. *The Traveling Salesman Problem*. Princeton, New Jersey: Princeton University, 2006. ISBN: 978-0-691-12993.

[9] Ke-Lin Du. *Search and Optimization by Metaheuristics*. Montreal, Canada: Birkhäuser, 2010. ISBN: 9783319411927.

[10] María V. García. «Problema del viajante de comercio (TSP) Métodos exactos de resolución». En: (2014), págs. 12-13.

[11] Hirst Jeffry L. Harris John M. et al. *Combinatorics and Graph Theory*. 233 Spring Street, New York, NY 10013, USA: Springer Science+Business Media, 2008. ISBN: 97803877977106.

[12] R.A. Hincapié Isaza et al. *Técnicas heurísticas aplicadas al problema del cartero viajante (TSP)*. 2005.

[13] G.F. Minetti. *Capítulo 5: Heurísticas Convencionales para TSP*. 2003, págs. 56-65.

[14] Bueno Guillén Silvia Núñez Valdés Juan Alfonso Pérez María. «Siete puentes, un camino: Königsberg». En: (2004), págs. 69-78.

[15] Alejandro Fuentes Penna. 2013.

[16] Coxeter H.S.M. Rouse Ball W.W. *Mathematical Recreations and essays*. Dover publications, 1987. ISBN: 9780486253572.

[17] Reyes Francisco Santamaría Carlos Cáceres Maciel. «Estudio Comparativo de Métodos de Solución al Problema del Agente Viajero para el Diseño de Rutas de Distribución en Empresa COMIX». En: (2015), págs. 24-30.

[18] Robert J. Vanderbei. *Linear Programming*. Princeton, New Jersey, USA: Springer, 2014. ISBN: 0884-8289.

[19] Wolfram. https://www.wolfram.com/mathematica/index.php.es?source=footer. Accessed: 2022-12-27. 2023.

[20] Victor Yepes. https://victoryepes.blogs.upv.es/2014/09/10/optimizacion-combinatoria/. Accessed: 2014-09-10. 2014.

Apéndice A

Código Python: solver CPLEX

Código para obtener las soluciones óptimas

```
from pyomo.environ import *
import numpy as np
import pandas as pd
import cplex

df = pd.read_excel('23.xlsx',header=None)
distancias = df.to_numpy()

n=len(distancias)

model=ConcreteModel()

model.cities=RangeSet(0,n-1)

model.x=Var(model.cities,model.cities,within=Binary)

model.t = Var(model.cities, within=NonNegativeIntegers,bounds=(0,n-1))
model.t[0].fix(0) #se fija t[0]=0

def fo_rule(model):
    return sum(distancias[i,j]*model.x[i,j]
               for i in model.cities for j in model.cities)
model.fo=Objective(rule=fo_rule,sense=minimize)

def goto_rule(model,i):
    return sum(model.x[i,j] for j in model.cities if i!=j)==1
model.goto=Constraint(model.cities,rule=goto_rule)

def comefrom_rule(model,j):
    return sum(model.x[i,j] for i in model.cities if i!=j)==1
model.comefrom=Constraint(model.cities,rule=comefrom_rule)

def mtz_rule(model,i,j):
    if i!=j and j>=1 :
        return model.t[j] >= model.t[i]+1-n*(1-model.x[i,j])
    else:
        return model.t[j] - model.t[j] ==0
model.mtz=Constraint(model.cities,model.cities,rule=mtz_rule)
```

```
# model.pprint()

# results = SolverFactory('cplex_direct').solve(model)
results = SolverFactory('cplex_direct').solve(model).write()
# model.pprint()
print('')
tour = [None] * n
for i in model.cities:
#     print('t[',i,']= ',value(model.t[i]))
    tour[int(value(model.t[i]))]=i
# print('')
# for i in model.cities:
#     for j in model.cities:
#         print('x[',i,',',j,']= ',value(model.x[i,j]))

print('Tour= ',tour)
print('fo= ',value(model.fo))
```

Código para graficar las soluciones óptimas

```
\textbf{Para obtener la gráfica de la solución:}

import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance

fig, ax = plt.subplots(2, sharex=True, sharey=True)         # Prepare 2 plots
ax[0].set_title('Raw nodes')
ax[1].set_title('Optimized tour')
ax[0].scatter(positions[:, 0], positions[:, 1])             # plot A
ax[1].scatter(positions[:, 0], positions[:, 1])             # plot B

for i, pos in enumerate(positions):
    ax[0].annotate(i, pos)

start_node = 0
tdistance = 0.
for i in range(n):
    start_pos = positions[start_node]
    if i!=n-1:
        next_node = tour[i+1]
        end_pos = positions[next_node]
        ax[1].annotate(tour[i+1],
            xy=start_pos, xycoords='data',
            xytext=end_pos, textcoords='data',
            arrowprops=dict(arrowstyle="<-",
                            connectionstyle="arc3"))
        tdistance += distancias[start_node,next_node]
        start_node = next_node
    else:
        start_pos = positions[tour[n-1]]
```

```
        end_pos = positions[0]
        ax[1].annotate(0,
        xy=start_pos, xycoords='data',
        xytext=end_pos, textcoords='data',
        arrowprops=dict(arrowstyle="<-",
                        connectionstyle="arc3"))
        tdistance += distancias[tour[n-1],0]


# textstr = "N nodes: %d\nTotal length: %.3f" % (N, tdistance)
# props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
# ax[1].text(0, -0.5 , textstr, transform=ax[1].transAxes, fontsize=14, # Textbox
#         verticalalignment='top', bbox=props)

print('N= ',n)
print('Distancia total= ',tdistance)
# plt.tight_layout()
plt.show()
```

Apéndice B

Wolfram Mathematica

```
coor = Import["C:/Users/spmx/Documents/ESPOL/Concorde/23.txt","Table"];
coord = Partition[Riffle[coor[[All, 2]], coor[[All, 3]]], 2];
dm = DistanceMatrix[coord];
newdiag = ConstantArray[9999, Dimensions[coord][[1]]];
dm = ReplacePart[dm, {i_, i_}: newdiag[[i]]];
Export["C:/Users/spmx/Documents/ESPOL/Concorde/23.xlsx",dm]
Export["C:/Users/spmx/Documents/ESPOL/Concorde/coord23.xlsx",coord]
```