DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Un algoritmo Branch-and-Cut para el AngTSP

Tesis de Licenciatura en Ciencias de la Computación

8 de marzo de 2013

Integrante	LU	Correo electrónico
Federico Javier Pousa	221/07	fpousa@dc.uba.ar

Directoras

Isabel Méndez-Díaz imendez@dc.uba.ar

Paula Zabala pzabala@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

A Papá y Mamá que hace 25 años que se vienen recibiendo.

A mis directoras, por todo el acompañamiento brindado para este trabajo.

A Isa, Pau, Juan y Pablo por darme un buen lugar y un buen ambiente para hacer esta tesis, y por responderme infinitas dudas.

A los jurados, por tomarse el tiempo para leer esta tesis y realizarle correcciones para mejorarla.

A Sol, Luján, Liliana, Delia, Fer, Ariel y toda mi familia, por apoyarme siempre.

A Facu, Brian y Doc por compatir toda la carrera y porque muchísimas cosas las aprendí de ustedes.

A todos mis compañeros de TPs, por formar parte de un pedazo muy grande del laburo hecho todos estos años.

A muchas personas que de una forma u otra fueron importante en mi carrera (Sepan que esta lista debería ser muuucho más larga): Pablic, Tincho, Agus Montero, Felipe, Kari, Martín, Dardo, Juli, Adri, Mariano, Pablo, Emanuel, Diego G, DFS, JPG, Mariano Moscato ...

A Agus, De, Tute, Maxi, Garyu, Tin y Chou por compartir casi todos los días de esta carrera.

A Rogelio, porque fue un compañero de estudio fundamental para lograr aclimatarme a esta carrera.

A Silvia, por toda la ayuda que nos dió estos años para poder ser felices juntos.

A Facu, Martu, Noe, Neru, Caro, Ceci y Ro por seguir bancándome después de todos estos años.

A Nico, Tin, Guada y Cami por ser parte de un año lleno de infinitas cosas nuevas.

A todos los chic@s de Física por ser mis primeros compañeros en la Facultad, especialmente a Dami, Neru y a sus familias por toda la hospitalidad para poder estudiar de la mejor manera.

A todo el DC por siempre invitarme a participar de las cosas interesantes que pasan acá dentro.

A todos mis docentes del DC por formarme durante estos años.

A mis alumnos del DC, porque muchas veces aprendo más de lo que les enseño.

A la escuela nro 21 y al Colegio Ward por ser parte de mi educación, con mención especial a Teresa Pizarri por ser la primera profesora que me introdujo un problema de optimización como tal.

A Fondo de Tabla por todas las cosas épicas, los 2 campeonatos ganados, el obelisco copado, los 11 torneos de futbol 5, los 2 torneos de futbol 11, las 3 rodillas, los esguinces, los desgarros, el 2 a 1 a Penetraitor, jugar con un bombo y no escuchar a mi defensor a un metro, tener más trapos que Arsenal, y muchas otras cosas más...

A Lisa y a Luna, por todas las lamidas.

A Mago de oz (y muuuuuchas otras músicas más), por la compañía en los tps, las juntadas, y los momentos previos a rendir algo importante. *We must be over the rainbow*

A Almendra, Sofia, Yoyiwo, Pongo y Perdyta, por todos sus clocks.

A Agus, por ser mi compañera inigualable en absolutamente todo, por el apoyo incondicional y por el amor en cada una de tus acciones. *Come what may*

Números destacados:

Ayudante con más presencias: David Gonzalez Marquez (Orga I, Orga II, Algo III)

Profesor con más presencias: Paula Zabala (Algo I, Algo III, Investigación Operativa)

Ranking Compañeros de tps:

Doc	19
Facu	19
Martin	8
Rogelio	7
Brian	6
Agus	5
Juan	3
Leopoldo	3
Dardo	2

Ranking Compañeros de finales:

Doc	7
Facu	4
Brian	3
Dami	2
Neru	2
Juli	2
Gabi	1
Marcos	1
Agus	1
Benito	1
Gaston	1
Felipe	1

Resumen

El objetivo del trabajo es desarrollar un algoritmo exacto para el Problema del Viajante de Comercio Angular basado en un modelo de Programación Lineal Entera.

El AngTSP o Problema de Viajante de Comercio Angular consiste en, dada una serie de puntos en el plano, encontrar un ciclo simple que pase por todos los puntos minimizando tanto la distancia recorrida, como la suma de los ángulos de giro realizados. La función objetivo de este problema tiene dos parámetros *alfa* y *beta* para indicar que importancia se le quiere dar a la minimización de las distancias, y que importancia se le quiere dar a la minimización de la suma de los ángulos.

El AngTSP encuentra sus aplicaciones en situaciones donde el ángulo de giro es costoso o prohibitivo, siendo el ejemplo más emblemático el diseño de trayectorias en robótica en donde el tiempo de giro no es despreciable respecto del tiempo total de recorrido.

Para diseñar un algoritmo para este problema se comenzó por estudiar los modelos presentes en la literatura para el Problema de Viajante de Comercio (TSP, *travelling salesman problem*), el cual es un caso particular del problema a tratar, que se obtiene cuando se le da total importancia a las distancias y ninguna importancia a los ángulos. Es por esta relación entre los problemas que es lógico creer que las formulaciones con buen desempeño para el TSP, también pueden tener buena performance en el AngTSP.

Se realizaron pruebas de los diferentes modelos sobre varias instancias, para finalmente elegir, según ciertos criterios, el mejor modelo entre los estudiados.

Una vez elegida la formulación preliminar se continuó con el diseño de un algoritmo branch-and-cut, realizándose los siguientes pasos:

- Se generaron dos heurísticias iniciales y se combinaron con dos metaheurísticas de búsqueda local para obtener cotas primales de calidad para comenzar la ejecución del branch-and-cut.
- Se realizó un estudio poliedral sobre la formulación para conseguir una más ajustada. Se conjeturó una caracterización de las igualdades del sistema minimal de la cápsula convexa de las soluciones factibles y se desarrollaron varias familias de desigualdades válidas con sus respectivos algoritmos de separación.
- Se diseñaron tres heurísticas primales con dos variaciones posibles para cada una.
- Se analizaron diferentes opciones para la selección de *branching* y la selección de nodo en el árbol de branch-and-cut.

En cada una de las secciones mencionadas se realizó un estudio cualitativo de las diferentes opciones barajadas para poder realizar una decisión sobre qué opción es más beneficiosa para el algoritmo.

Por último, se evaluó la performance del algoritmo en diferentes instancias que muestran que la técnica propuesta es eficiente.

Índice general

Capítulo 1

Introducción al AngTSP

El Problema del Viajante de Comercio Angular(AngTSP) consiste en encontrar un camino entre un conjunto de puntos, de forma que se visiten todos los puntos disponibles una única vez, comenzando y terminando en el mismo lugar, y buscando minimizar no sólo la distancia recorrida en todo el camino, si no también la suma de los ángulos que se giró en todo el recorrido.

El problema fue introducido en [2] motivado por las aplicaciones en robótica para diseñar trayectorias para ciertos tipos de robots. Primero al problema se lo introduce en su versión puramente angular, lo que significa que sólo se tiene como objetivo minimizar la suma de los ángulos del circuito sin tener en cuenta las distancias entre los puntos. Luego, en ese mismo trabajo, se presenta la ampliación en donde el objetivo es un compromiso entre minimizar la distancia y el ángulo total.

Para dos ejes (u,v) y (v,w) incidentes a un mismo vértice v, se define el ángulo entre esos ejes como el valor absoluto del cambio de dirección en el movimiento de viajar de u a w pasando por v. Formalmente, el problema se define como encontrar un tour, o ciclo simple, de un conjunto de puntos en un espacio Euclideo con ángulo total mínimo.

El problema está motivado, lógicamente, por aplicaciones en donde es importante encontrar un tour para viajar por todos los puntos pero donde el ángulo de giro no es despreciable, ya sea bien porque el tiempo para el cambio de dirección es considerable, o bien porque el actor encargado de realizar el tour posee limitaciones físicas sobre los giros que puede realizar.

En particular, las aplicaciones directas de este problema, se encuentran en el campo de la robótica en donde, como se mencionó antes, el cambio de dirección es un aspecto a considerar en los tiempos finales de recorrido del tour. Por otro lado, también existen aplicaciones donde los giros tienen limitaciones físicas como por ejemplo en el desarrollo del entramado de las vías ferroviarias donde, lógicamente, no se puede pretender que el ferrocarril gire en cualquier ángulo en cualquier punto.

En [2] el problema es solamente estudiado desde el punto de vista teórico, se analiza y se demuestra que el mismo pertenece a la clase de problemas NP-Difícil, así como también se demuestra que el problema puede ser aproximado a razón de $\mathcal{O}(\log n)$ en tiempo polinomial.

Luego, en [17] el problema se estudia desde un costado práctico. En ese trabajo se busca resolver el problema del viajante de comercio para vehículos Dubins[20], que son vehículos que sólo pueden describir trayectorias $\mathcal{C}^2$ (curvas que poseen primera y segunda derivada, y ambas son continuas) y de limitada curvatura. Para resolver el problema mencionado, los autores dividen el problema en dos fases. La primera fase, o *tour stage* según los autores, se basa en determinar el orden en el que van a ser visitados los puntos. La segunda fase, o *trajectory stage*, consiste en determinar la trayectoria entre cada uno de los puntos.

Es para la mencionada primera fase que los autores del trabajo proponen la utilización del AngTSP y obtienen interesantes resultados.

Hasta donde llega nuestro conocimiento, sólo se encontraron estos dos trabajos con referencias al Problema de Viajante de Comercio Angular, siendo [17] el único que lo lleva a la práctica, pero lo hacen solamente adaptando un modelo del TSP y entregándoselo directamente a un software de resolución general sin darle mayor estudio particular al mismo.

En este trabajo se analiza la versión del problema que tiene en cuenta tanto la distancia total, como la suma de los ángulos. Lo que se busca optimizar es la suma de estos dos aspectos, dándole diferentes pesos a cada uno de ellos dependiendo de si se quieren soluciones donde el giro total sea menor, probablemente en detrimento de la distancia total recorrida, o soluciones donde lo más importante sea la distancia de viaje, y en menor medida los grados de giros realizados.

Es decir, la función objetivo es de la forma

$$\alpha \times distanciaTotal + \beta \times anguloTotal \tag{1.1}$$

El α y el β son justamente los parámetros para poder pesar cada uno de los aspectos. La fijación de estos parámetros depende de la aplicación particular en la que se necesite resolver el problema.

Por ejemplo, en la siguiente figura se puede ver un conjunto de puntos en el plano que se quieren recorrer.

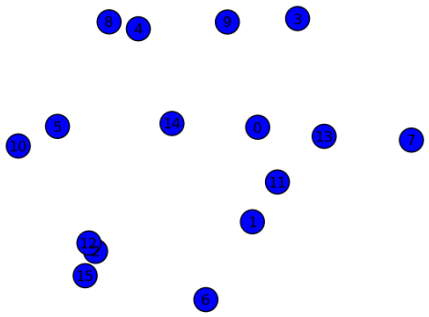

Figura 1.1: Puntos a visitar

La siguiente figura es el mejor recorrido si sólo se tienen en cuenta las distancias y no los ángulos, es decir fijar el parámetro α en 1 y β en 0.

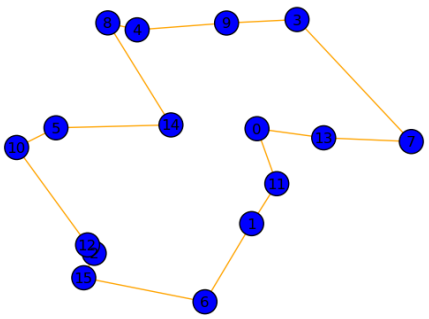

Figura 1.2: Solución minimizando distancias

El siguiente tour muestra cual es la mejor solución si no importan las distancias y sólo se minimiza la suma de los ángulos, lo que implica α en 0 y β en 1.

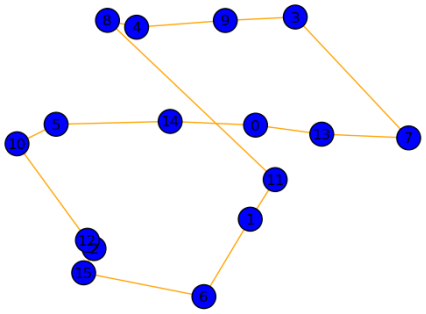

Figura 1.3: Solución minimizando ángulos

Se puede ver como en la solución para ángulos se recorren todos los puntos que están a media altura juntos, siguiendo casi una recta. Mientras que en la solución para minimizar las distancias se prefiere recorrer los puntos del lado derecho por un lado, y los del lado izquierdo por el otro.

Si bien la fijación de los parámetros puede ser completamente *ad hoc* dependiendo del problema o la necesidad del usuario, existen algunas situaciones en donde se encuentra estudiada cuál es la configuración óptima de los mismos. Por ejemplo, en [13] y [17] se menciona el hecho de que se puede demostrar cual es la combinación de parámetros para conseguir el tour con mínimo tiempo de viaje para robots de tipo *differential-drive*.

Los robots de tipo *differential-drive*, poseen dos ruedas con la posibilidad de rotar a diferentes velocidades para poder realizar giros. En los trabajos citados se menciona que el tiempo mínimo de viaje se consigue fijando α en 1 y β en ρ, siendo 2ρ la distancia que separa las dos ruedas del robot.

El objetivo de este trabajo es desarrollar un algoritmo exacto para el Problema del Viajante de Comercio Angular utilizando modelos de Programación Lineal Entera.

Luego de realizar una introducción a la Programación Lineal Entera, el siguiente capítulo se dedica a realizar un estudio cualitativo de las diferentes formulaciones conocidas para el Problema de Viajante de Comercio y su adaptación a la versión angular. De esta forma se selecciona una formulación inicial para su posterior fortalecimiento.

Luego, se dedica un capítulo al estudio de heurísticas iniciales para conseguir soluciones factibles para el problema. En este capítulo se presentan dos heurísticas iniciales y se combinan con dos metaheurísticas de búsqueda local para obtener cotas primales de calidad.

En el siguiente capítulo se desarrolla el estudio poliedral realizado sobre la formulación para conseguir una versión más ajustada. Se caracterizan las igualdades del sistema minimal y se desarrollan varias familias de desigualdades válidas con sus respectivos algoritmos de separación.

Una vez finalizado el estudio poliedral, se presenta un capítulo que se centra en el estudio de diferentes heurísticas primales y se estudia la diferencia de performance al utilizar diferentes estrategias de generación y recorrido del árbol, tanto durante el momento del *branching*, como en la selección del próximo nodo a resolver.

Por último se presentan los resultados obtenidos sobre varias instancias, para poder comparar la performance del algoritmo original contra la del algoritmo luego de todas las modificaciones diseñadas.

Capítulo 2

Introducción a la Programación Lineal Entera

2.1. Definición

La Programación Lineal es una técnica para resolver problemas de optimización. Mediante una formulación de Programación Lineal se pueden modelar situaciones donde se debe minimizar una función lineal, denominada función objetivo, sujeta a un conjunto de restricciones, también lineales. Todo problema de Programación Lineal es resoluble de manera eficiente [8], pero existen muchos problemas que no se pueden modelar con esta técnica a menos que se restrinja el dominio de las variables.

La Programación Lineal Entera está intimamente relacionada con la Programación Lineal, pero tiene la diferencia de que todas las variables están restringidas a tomar valores enteros.

La Programación Lineal Entera Mixta es una generalización de las dos anteriores, en donde algunas o todas las variables estás restringidas a valores enteros.

La Programación Lineal Entera Mixta no sólo puede resolver los mismos problemas que resuelve la Programación Lineal, por definición, sino que también puede modelar problemas donde las variables son enteras. Esto por un lado se traduce en problemas donde no se pueden tener valores reales, por ejemplo en los problemas de producción de unidades indivisibles; pero también se traduce en modelos mucho más potentes, ya que la presencia de variables enteras o binarias permiten la modelización de restricciones lógicas como conjunciones, disyunciones y consecuencias. El caso más emblemático es el problema SAT, que puede ser resuelto mediante Programación Lineal Entera Mixta y no mediante Programación Lineal.

Esta diferencia en los problemas que se pueden modelar, hace que la Programación Lineal Entera Mixta sea NP-Difícil en general. Por lo que no existe ningún algoritmo polinomial que resuelva cualquier problema de Programación Lineal Entera Mixta a menos que P = NP. Una demostración de que la Programación Lineal Entera Mixta es NP-Difícil se obtiene justamente mostrando que SAT se puede reducir a un problema de Programación Lineal Entera Mixta.

La Programación Lineal Entera Mixta puede ser utilizada para resolver variados problemas de optimización, entre los que se encuentran problemas de ruteo de vehículos, problemas de telecomunicaciones, problemas de planificación de producción, problemas de asignación de tareas y turnos, entre otros.

En las próximas secciones se detallan algunos conceptos básicos de la Programación Lineal Entera Mixta y se presentan los algoritmos generales de resolución conocidos. Para una ampliación de los temas aquí introducidos ver [21].

2.2. Formulaciones

Un problema de Programación Lineal Entera Mixta(PEM) puede ser formulado de la siguiente manera:

$$
\begin{aligned}
\textit{Minimizar} \quad & \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \\
\textit{sujeto a} \quad & \\
& \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \quad i = 1, \ldots, m \\
& x_j \in \mathbb{Z}_+ \qquad \forall j \in I \\
& x_j \in \mathbb{R}_+ \qquad \forall j \in C
\end{aligned}
$$

donde I es el conjunto de variables enteras, C es el conjunto de variables continuas y m es la cantidad de restricciones.

Cada formulación PEM tiene asociado un poliedro $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ con $A \in \mathbb{R}^{mxn}$ y $b \in \mathbb{R}^m$ y el conjunto de soluciones factibles $S = P \cap \{x \in \mathbb{R}^n : x_j \in \mathbb{Z}\ \forall j \in I\}$. A P se lo denomina relajación lineal de S.

Cada desigualdad presentada en las restricciones divide el espacio donde viven las soluciones en dos subespacios. El subespacio que será tenido en cuenta para definir los puntos factibles dependerá de la orientación de la desigualdad que se tenga que cumplir.

A continuación se presenta el poliedro asociado a una formulación con sólo dos variables, y ambas restringidas a ser enteras.

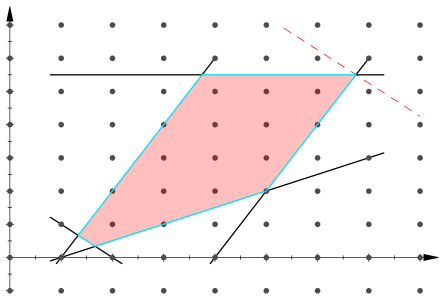

Figura 2.1: Conjunto factible y poliedro asociado

Las líneas que definen el poliedro son las desigualdades presentes en la formulación, y el área encerrada por estas líneas resulta en el poliedro asociado gracias a la orientación de las desigualdades. Si bien este poliedro contiene muchos puntos, el conjunto de soluciones factibles S son los puntos enteros dentro de P.

La línea que se observa punteada corresponde a la función objetivo. Al ser esta una función lineal, en el espacio queda representada por un hiperplano, en este caso una línea, que sólo tiene definida su pendiente ya que no tiene término independiente como las desigualdades de las restricciones. Es decir, la función objetivo es un hiperplano con pendiente definida que toma diferentes valores según donde esté ubicada. Lo buscado es que la función objetivo pase por el punto del conjunto factible de soluciones que mayor valor le haga tomar. En el caso de la figura la función objetivo se encuentra optimizada para todos los puntos del poliedro P, y la mejor solución se encuentra en la esquina superior derecha.

En los problemas de Programación Lineal las soluciones factibles son todos los puntos del poliedro ya que no está la restricción de integralidad. Esto hace que en un poliedro acotado, como el mostrado en la figura, se pueda demostrar que la mejor solución está en uno de los puntos extremos o esquinas del mismo. Por lo que en un problema de Programación Lineal, se pueden recorrer las esquinas hasta encontrar la que tenga menor valor objetivo, siempre haciendo referencia a una minimización.

Si llamamos $conv(S)$ a la cápsula convexa de S (menor poliedro que contiene a S), entonces PEM es equivalente a resolver $Min\ cx : x \in conv(S)$. Si $P = conv(S)$, el problema PEM puede ser resuelto en forma eficiente por cualquier algoritmo de Programación Lineal.

Esto sucede porque un algoritmo para resolver Programación Lineal puede encontrar el mínimo en todo el poliedro P y, por lo dicho anteriormente, este mínimo será alguna de las esquinas del poliedro. Como en la cápsula convexa las esquinas coinciden con puntos enteros, entonces cuando se minimice el problema sobre todo P, resultará que el mínimo es un punto entero, por lo que este punto es mejor, en términos de la función objetivo, que cualquier otro punto de P y en particular que cualquier otro punto de S, ya que S $\subseteq$ P.

Si se conociese la descripción de $conv(S)$ mediante un número polinomial (en la cantidad de variables) de desigualdades lineales, se podría resolver el problema como uno de Programación Lineal, lo cual es computacionalmente fácil. Es más, aún en el caso que esta caracterización no fuese polinomial, bajo ciertas circunstancias el problema podría ser resuelto en tiempo polinomial. Desafortunadamente, para la mayoría de los problemas no se ha podido obtener la descripción completa de la cápsula convexa y, en general, el número de restricciones lineales que la caracterizan es exponencial.

Para un mismo problema existen muchas formas de formularlo, a veces con las mismas variables existen otras restricciones que también modelan el problema, pero que devienen lógicamente en poliedros diferentes. Otras veces un problema se puede formular con dos conjuntos de variables diferentes, lo que hace que problablemente los poliedros asociados ni siquiera pertenezcan a la misma dimensión.

En la siguiente figura se pueden ver los poliedros asociados a dos formulaciones diferentes, siempre manteniéndose en dos dimensiones para poder visualizarlo correctamente.

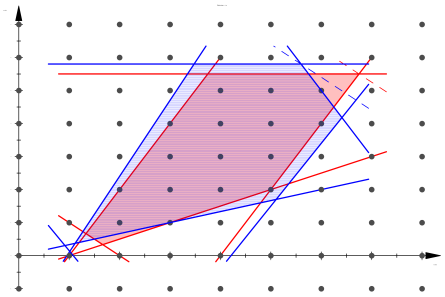

Figura 2.2: Dos formulaciones y sus poliedros asociados

Por un lado se ve el poliedro rojo opaco, y por el otro el poliedro azul a rayas. Lo que se puede ver es que el conjunto de puntos enteros S es el mismo, pero el poliedro asociado P es diferente.

Se dice que una formulación $F1$ es *más ajustada* que otra, $F2$, cuando el poliedro asociado a $F1$ está incluido en el poliedro asociado a $F2$. En este caso $F1$ logra capturar todos los puntos enteros posibles y contiene menos puntos fraccionarios obsoletos que $F2$.

Esta noción conforma una relación de orden que no es total. En la figura anterior se ve que hay sectores coloreados de rojo, pero no de azul, y viceversa. Esto indica que ninguna de las dos formulaciones es más ajustada que la otra.

Bajo esta definición, la cápsula convexa de un conjunto de soluciones factibles representa la formulación más ajustada que se puede conseguir. En la siguiente figura se muestra la primera formulación visualizada, junto con la cápsula convexa de los puntos de S. En rojo opaco se encuentra la primera formulación, y con la línea gruesa azul se remarca la cápsula convexa.

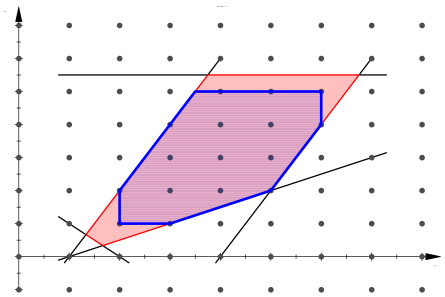

Figura 2.3: Formulación y Cápsula convexa

El objetivo para un problema de Programación Lineal Entera es diseñar las formulaciones más ajustadas posibles. Esta tarea en general resulta difícil, dado que la ayuda gráfica sólo sirve para los problemas en dos o tres dimensiones, cuando muchos de los problemas resueltos suelen tener cientos de miles de variables.

2.3. Algoritmos para Problemas de Programación Lineal Entera

El procedimiento más simple para resolver un problema de Programación Lineal Entera pura es enumerar todas las posibilidades. Sin embargo, debido a la explosión combinatoria esta técnica sólo resulta aplicable a instancias sumamente pequeñas. En esta sección se describen los algoritmos más usados en la práctica.

Los algoritmos más utilizados se encuadran dentro de algunos de estos esquemas básicos:

- Enumeración inteligente: algoritmos *Branch-and-Bound*.
- Caracterización de $conv(S)$ o ajuste de la relajación lineal: algoritmos de planos de corte.
- Una combinación de las dos técnicas anteriores: algoritmos *Branch-and-Cut*.

A continuación se describen los puntos más sobresalientes de cada uno.

2.3.1. Algoritmos branch-and-bound

Ya se mencionó que la enumeración de las soluciones factibles en busca de la solución óptima no es un procedimiento aconsejable para usar en la práctica. Para mejorar esta técnica básica muchas veces es posible eliminar algunas posibilidades mediante argumentos de dominancia o factibilidad. Es decir, argumentos que permiten afirmar que el óptimo no pertenece a un determinado subconjunto de soluciones sin la necesidad de enumerarlo.

Dentro de esta línea, en los años 60 fueron propuestos los algoritmos *Branch-and-Bound*, donde el *branching* se refiere a la parte enumerativa y el *bounding* al proceso de poda de posibles soluciones.

Estos algoritmos están asociados al concepto *divide y conquista*: si resulta difícil buscar el óptimo en un conjunto S, entonces es mejor buscar en partes de S y luego quedarse con la mejor

solución.

Este esquema puede ser representado mediante un árbol cuya raíz corresponde al problema original y sus ramas resultan de la división en partes del espacio de búsqueda. A cada nodo del árbol le corresponde un subproblema que consiste en buscar el óptimo en una parte del espacio de soluciones. Los argumentos de dominancia y factiblidad son los que permitirán descartar ramas del árbol en el proceso de búsqueda.

Una forma de llevar a cabo la poda, *bounding*, es calcular en los nodos del árbol cotas inferiores del óptimo del problema restringido a esa parte del espacio de soluciones. Si la cota es peor que la mejor solución obtenida hasta el momento, no es necesario explorar dicha parte. El cálculo de estas cotas debe lograr un equilibrio entre calidad y esfuerzo en obtenerla. Una cota débil hará que se explore innecesariamente ramas del árbol, pero un procedimiento que brinde buenas cotas a un costo alto puede no justificarse.

Para obtener cotas inferiores, una posibilidad es relajar el problema de forma de obtener una relajación *fácil* de resolver. La idea es reemplazar un PEM difícil por un problema de optimización más simple cuyo valor óptimo sea menor o igual al óptimo del problema original. Obviamente, es deseable obtener relajaciones *ajustadas*, es decir, que la diferencia relativa (*gap*) entre el valor óptimo de la relajación y el valor óptimo del PEM sea chica. Hay dos posibilidades obvias para que el problema relajado tenga esta característica. Se puede agrandar el conjunto de soluciones factibles sobre el cual se optimiza o reemplazar la función objetivo por otra que tenga igual o menor óptimo. Dentro de la primera posibilidad se encuentra la relajación lineal y en la segunda se enmarca la relajación lagrangeana. Las relajaciones no sólo son útiles para obtener cotas inferiores, algunas veces permiten probar optimalidad.

La relajación lineal consiste en borrar del PEM la imposición de ser entera sobre las variables que la tengan, es lo que en los gráficos mostrados anteriormente se enunciaba como el poliedro P asociado a S. Es la relajación más natural y una de las más utilizadas. La relajación lagrangeana consiste en remover un subconjunto de las restricciones que no incluya las restricciones de no negatividad. La violación de las restricciones relajadas es penalizada incluyendo estas restricciones, con un multiplicador no negativo, en la función objetivo. Los multiplicadores son iterativamente actualizados para maximizar la cota inferior obtenida del problema relajado.

Esencialmente, hay dos factores decisivos en la implementación de un algoritmo de este tipo: las reglas de *branching* y el esquema de selección del próximo nodo a explorar. No hay una combinación de estos factores que resulte mejor para todos los problemas. Es necesario utilizar criterios basados en una combinación de teoría, sentido común y experimentación.

El proceso de *branching* consiste en dividir la región factible anterior en dos o más regiones factibles más pequeñas. Cada nueva región da origen a un nuevo subproblema o nodo hijo, originado por la adición de una nueva restricción al problema del nodo padre. Un requerimiento esencial es que cada solución entera factible del nodo padre pertenezca a, al menos, uno de los hijos. Estos nuevos subproblemas son agregados a la lista de nodos activos, es decir, aún no explorados. La regla de *branching* más simple consiste en considerar alguna variable entera que tiene valor fraccionario, d, en la solución actual. Se parte al problema en dos hijos, imponiendo en uno de ellos como cota superior de este variable el valor $\lfloor d \rfloor$ y en el otro como cota inferior $\lceil d \rceil$. Este procedimiento es aplicado recursivamente a cada nodo del árbol.

En la siguiente figura se puede ver como la formulación original es partida en dos subproblemas, tomando la variable x y separando cuando vale menor o igual a 3, y cuando vale mayor o igual a 4. Se puede ver que esta separación propuesta cumple con el requisito de que toda solución entera factible se encuentra en alguno de los dos subproblemas nuevos.

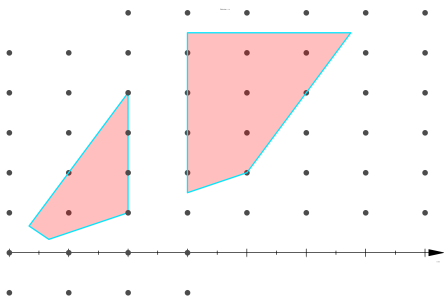

Figura 2.4: Regla de Branching

Es importante notar que la idea de esta simple regla de branching tiene por objetivo también que el punto fraccionario óptimo actual no se encuentre en ninguno de los dos subproblemas. Tomando la imagen anterior, si el óptimo de la relajación lineal hubiese estado en la esquina superior izquierda del poliedro donde la variable x vale entre 3 y 4, el branching propuesto lograría dejar de tener en cuenta este punto.

La próxima decisión que se debe tomar es la selección del siguiente nodo a explorar de la lista de nodos activos. En la práctica hay varios argumentos contradictorios que pueden ser utilizados. Como sólo es posible podar significativamente el árbol de enumeración si se cuenta con buenas cotas superiores, entonces se debería descender lo más pronto posible en el árbol para encontrar rápidamente soluciones factibles. Esto sugiere el uso de una estrategia de *búsqueda en profundidad*. Otra estrategia sugiere elegir el nodo activo con mejor cota (más chica). De esta manera, nunca se dividiría un nodo con cota inferior mayor que el valor óptimo del problema. Esta estrategia es llamada *mejor cota primero*.

El esquema básico del algoritmo es el siguiente. Llámese PEM el problema entero mixto que se quiere resolver, $\mathcal{N}$ al conjunto de subproblemas o nodos del árbol de enumeración activos. Para cada nodo k, $PL(k)$ representa la relajación lineal del PEM asociado a este nodo y Z^k el valor óptimo de $PL(k)$. En Z^* se almacena el valor de la mejor solución obtenida.

---

1. **Inicialización:**
   $\mathcal{N} = \{\text{PEM}\} \quad Z^* = \infty$

2. **Elección de próximo nodo:**
   Si $\mathcal{N} = \{\}$ el algoritmo termina. Si $Z^* \neq \infty$ entonces es óptimo. Si no, PEM es no factible
   Si $\mathcal{N} \neq \{\}$, elegir y borrar un nodo k de $\mathcal{N}$

3. **Evaluación:**
   Resolver $PL(k)$.

   a) Si no es factible, ir a **Elección**.

   b) *Bound*: si $Z^k > Z^*$, ir a **Elección**.

   c) Si la solución óptima cumple las condiciones de integralidad, actualizar $Z^* = \text{mín}\{Z^*, Z^k\}$ e ir a **Elección**.

4. **División:** (*Branch*) Particionar la región factible de $PL(k)$ en dos o más regiones, agregando un nuevo nodo a $\mathcal{N}$ por cada nueva región. Ir a **Elección**.

2.3.2. Algoritmos de planos de corte

Los algoritmos de planos de corte fueron originalmente propuestos por Gomory en los 60's [5] como un método general para resolver problemas de programación lineal entera.

Un algoritmo básico de planos de corte en un primer paso relaja las condiciones de integralidad sobre las variables y resuelve el programa lineal resultante. Si el programa lineal es infactible, el programa entero también lo es. Si la solución óptima del programa lineal cumple las condiciones de integralidad, se ha encontrado un óptimo del problema. En caso contrario, se busca identificar desigualdades lineales (*problema de separación*) que estén violadas por la solución fraccionaria del programa lineal y sean válidas para los puntos enteros factibles. Es decir, desigualdades que *separen* el óptimo fraccionario de $conv(S)$.

Siguiendo con el ejemplo gráfico que se presentó anteriormente, se puede visualizar la idea de este método. En la siguiente figura se muestra la formulación original con su óptimo en la relajación lineal, y una desigualdad válida que fortalece al modelo y deja afuera al anterior óptimo ya que era un punto fraccionario que no se quiere tener en cuenta.

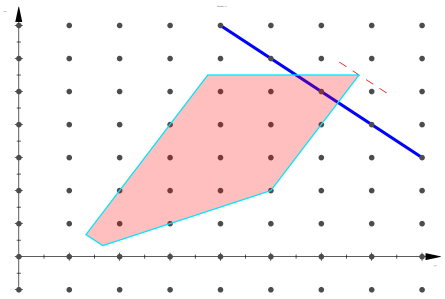

Figura 2.5: Plano de corte

Se observa como la línea gruesa azul corta la solución fraccionaria óptima que existía hasta el momento, pero no corta ningún punto entero factible. De esta manera, si se vuelve a resolver la relajación lineal, pero ahora con el modelo fortalecido, dará un óptimo diferente.

El algoritmo continúa hasta que:

- una solución óptima entera es encontrada, en cuyo caso el problema es resuelto con éxito
- el programa lineal es infactible, lo que significa que el problema entero es infactible
- no se pudo identificar alguna desigualdad lineal que separe al óptimo de la relajación actual, ya sea porque no se conoce la descripción completa de la cápsula convexa o porque los algoritmos de separación no son exactos.

El éxito del algoritmo depende en gran medida de la posibilidad y la eficiencia de encontrar desigualdades violadas (*planos de corte*) que puedan ser agregadas a la formulación para separar las soluciones fraccionarias.

Los planos de corte pueden ser generados bajo dos enfoques:

- *Con herramientas generales aplicables a cualquier problema de programación lineal entera*

  El algoritmo original propuesto por Gomory utiliza como planos de corte desigualdades derivadas del *tableau* óptimo de la relajación lineal, llamados *cortes de Gomory*. Aunque fue demostrado que este algoritmo, bajo ciertas condiciones, termina en un número finito de pasos, en la práctica su convergencia parece ser muy lenta. Por otro lado, la implementación

computacional es numéricamente inestable, aunque en la actualidad han sido fortalecidos lográndose buenas implementaciones.

Posteriormente, se han desarrollado algoritmos que utilizan una variedad de cortes aplicables a cualquier PEM, como por ejemplo los cortes *disyuntivos, clique, cover*, etc. Si bien estos algoritmos tienen propiedades teóricas de mucho interés, su éxito en la práctica es discutible. Cualquiera de las técnicas mencionadas tienen la ventaja de poder ser utilizadas para cualquier problema de Programación Lineal Entera, independientemente de su estructura. Si bien esto es una propiedad deseable en un algoritmo, no siempre brinda la herramienta más adecuada para casos particulares. Un estudio más específico del problema ayuda a obtener mejores procedimientos. Este es el sentido del próximo enfoque.

- *Explotando la estructura particular del problema.*

  En los 70's, resurgió el interés por los algoritmos de planos de corte debido al desarrollo de la teoría poliedral. Mediante el estudio de combinatoria poliedral, la intención es reemplazar el conjunto de restricciones de un programa de programación entera mixta por la descripción de la cápsula convexa del conjunto de soluciones factibles. Las desigualdades lineales necesarias para describir a $conv(S)$ se denominan *facetas*. Si se conoce de forma completa esta descripción, el problema entero puedo ser resuelto como un problema de programación lineal. De esta manera, explotando la estructura particular de cada problema, los planos de corte resultarán más efectivos a la hora de cortar soluciones.

  Desafortunadamente, no es fácil tener esta descripción y los problemas pertenecientes a la clase NP-Difícil tienen una cantidad exponencial de facetas, a menos que $P = NP$. Aún en ese caso, la cantidad de facetas podría ser exponencial, pero el problema de separación de las mismas sería polinomial. Alternativamente, es posible utilizar cualquier desigualdad válida para el conjunto de soluciones factibles como planos de corte, pero, en general, la eficiencia del algoritmo depende de la *fortaleza* de estas desigualdades, siendo las facetas los *mejores* cortes posibles.

  Con fines algorítmicos, el estudio poliedral debe estar acompañado de algoritmos de separación eficientes. En este sentido, hay un resultado muy importante debido a Grötschel, Lovász y Schrijver [8] que relaciona la complejidad del problema de separación con la complejidad del problema de optimización. Se establece que el problema de optimización $max\{cx : x \in conv(S)\}$ puede resolverse polinomialmente si y sólo si el problema de separación ($x \in conv(S)$ *ó encontrar una desigualdad válida violada*) es polinomial. Es decir que si el problema que se está tratando no es polinomial, existe al menos alguna familia de facetas que no puede separarse en tiempo polinomial. Esto de alguna manera implica el grado de dificultad de encontrar la descripción de todas las facetas de la cápsula convexa y del desarrollo de algoritmos de separación.

En forma general, para desarrollar un algoritmo de planos de corte, primero se busca una descripción parcial de la cápsula convexa del conjunto de las soluciones factibles enteras o desigualdades válidas *fuertes* para este conjunto. Luego es necesario el diseño de rutinas de separación para las familias de desigualdades encontradas. Estas rutinas toman como entrada una solución y retornan restricciones de estas familias violadas por este punto, si es que existe alguna. El problema de separación, en algunos casos, puede ser NP-difícil o tener complejidad alta, lo que lleva en la práctica a utilizar algoritmos heurísticos, o sea, que es posible que no sean capaces de encontrar una desigualdad violada aunque exista. La estrategia que se utilice para decidir la búsqueda en la diferentes familias es clave para la performance del algoritmo.

El esquema básico de un algoritmo de planos de corte es el siguiente. Llamese PEM al problema entero mixto que se quiere resolver, $PL(P)$ a la relajación lineal del problema P y x^P la solución óptima de esta relajación.

---

1. **Inicialización:**
   $P =$ PEM
2. **Evaluación:**
   Resolver $PL(P)$
   a) Si es no factible, entonces PEM es no factible y el algoritmo termina.
   b) Si x^P cumple las condiciones de integralidad, x^P es la solución óptima de PEM y el algoritmo termina.
   c) **Separación:** Caso contrario, resolver el problema de separación para x^P.
   - Si se encuentran cortes, agregarlos a P e ir a **Evaluación**.
   - Caso contrario, retornar el funcional evaluado en x^P como una cota inferior de PEM.

El algoritmo de planos de corte puede no resolver el problema de forma óptima, ya sea por no encontrar desigualdades válidas violadas o porque el tiempo consumido excede el tiempo disponible. Sin embargo, puede ser utilizado para generar buenas cotas inferiores del valor óptimo del problema. Además, muchas veces a partir de la solución óptima de la relajación actual es posible encontar buenas soluciones enteras mediante una heurística, brindando una cota superior del valor óptimo.

2.3.3. Algoritmos branch-and-cut

En muchas instancias, los dos algoritmos descriptos arriba fallan en la resolución del problema. A comienzos de los 80's se comenzó a aplicar una metodología mixta que conjuga las dos ideas dando origen a los llamados algoritmos *Branch-and-Cut*. De esta manera se lograron resolver exitosamente instancias de tamaño considerable de una gran cantidad de problemas de programación lineal entera, como por ejemplo el *Problema de Viajante de Comercio* [19], el *Problema de Ordenamiento Lineal* [6], el *Problema de Corte Máximo* [7], etc.

Uno de los factores que influye en el fracaso de los algoritmos *Branch-and-Bound* es la baja calidad de las cotas obtenidas mediante las relajaciones lineales. Esto significa que resulta crucial poder ajustar las formulaciones, por ejemplo con planos de corte.

Un algoritmo *Branch-and-Cut* es un *Branch-and-Bound* en el cual se generan planos de corte a través del árbol de enumeración. El objetivo de esto es reducir significativamente el número de nodos del árbol mejorando la formulación de los subproblemas. En un *Branch-and-Cut*, la enumeración y los planos de corte se benefician mutuamente. Generalmente, la cota producida en cada nodo del árbol de enumeración es mejor que en un *Branch-and-Bound*, debido a las nuevas desigualdades agregadas a la formulación del correspondiente subproblema. Por otro lado, el proceso de *branching* perturba la solución fraccionaria ayudando a los algoritmos de separación.

Estos algoritmos no sólo son capaces de producir la solución óptima, también pueden brindar soluciones aproximadas al óptimo con certificado de calidad en tiempos de cómputo moderado. Es decir, aún sin conocer el valor real del óptimo, se puede acotar que tan lejos se está de este.

En la implementación de un algoritmo *Branch-and-Cut* hay que tener en cuenta las estrategias propias de un algoritmo *Branch-and-Bound* sumadas a las de un algoritmo de planos de corte. Además, se agregan nuevas decisiones como ¿cuándo buscar planos de cortes?, ¿cuántos cortes agregar?, etc.

El esquema de un algoritmo *Branch-and-Cut* es el siguiente.

---

1. **Inicialización:**
   $\mathcal{N} = \{\text{PEM}\} \quad Z^* = \infty$
2. **Elección de próximo nodo:**
   Si $\mathcal{N} = \{\}$ Z^* es óptimo y el algoritmo termina
   Si no, elegir y borrar un nodo k de $\mathcal{N}$
3. **Evaluación:**
   Resolver $PL(k)$.
   a) Si es no factible, ir a **Elección**.
   b) Si $Z^k > Z^*$, ir a **Elección**.
   c) Si la solución óptima cumple las condiciones de integralidad, poner $Z^* = \text{mín}\{Z^*, Z^k\}$ e ir a **Elección**.
4. **División vs Separación:**
   Decidir si se buscarán planos de corte:
   - **SI:** Ir a **Separación**
   - **NO:** Ir a **División**
5. **División:** Particionar la región factible de PL(k) en dos o más regiones, agregando un nuevo nodo a $\mathcal{N}$ por cada nueva región. Ir a **Elección**.
6. **Separación:** Resolver el problema de separación para la solución fraccionaria de $PL(k)$.
   - Si son encontrados cortes, agregarlos a la formulación e ir a **Evaluación**.
   - Si no se encuentran, ir a **División**.

Capítulo 3

Formulaciones para el AngTsp

El Problema del Viajante de Comercio es un problema muy abordado en la literatura, teniendo más de 50 años de estudio, tanto en sus formulaciones matemáticas, como en los resultados computacionales obtenidos al implementar las distintas formulaciones. En [4] se puede observar uno de los trabajos fundacionales sobre este problema.

Dada las similitudes entre el TSP y el AngTSP, el objetivo es realizar un estudio preliminar de varios modelos para el TSP presentes en la literatura, y su adaptación al AngTSP, para poder elegir una buena formulación para su posterior análisis en profundidad.

3.1. Formulaciones para el TSP

Existen muchas formulaciones para el Problema del Viajante de Comercio que se pueden encontrar en la literatura, se puede consultar [18] como una buena referencia para todas las formulaciones que serán presentadas en este trabajo.

Dichas formulaciones se encuentran usualmente divididas por las abstracciones que generan las diferentes restricciones. Entre las abstracciones más usuales se encuentran los modelos con *rompimiento de subtours*, los basados en *commodities flow*, los basados en *variables de precedencia* y los basados en *dependencias temporales*. Todas estas ideas se irán desarrollando en cada una de las formulaciones que se explican en esta sección.

A continuación, se presentan los modelos clásicos del TSP que luego serán extendidos a la variante angular, para poder realizar un estudio cualitativo de las diferentes opciones.

En todas las formulaciones se trabaja sobre un grafo G, que tiene como conjunto de vértices a los puntos en el plano que se quieren visitar, y donde un eje representa el camino de un punto hasta el otro, teniendo asociado un peso que es la distancia que existe entre ellos. A priori no se asume que las distancias entre dos puntos sea simétrica, ya que pueden existir instancias donde la simetría no se cumpla, por lo que siempre existen dos ejes entre cada par de nodos, uno en cada dirección.

3.1.1. Modelo de Dantzig, Fulkerson y Johnson(DFJ)

Este modelo usa variables x_{ij} para indicar la utilización o no en el tour del eje entre los nodos i y j del grafo. Formalmente, x_{ij} es 0 si y sólo si el tour no contiene el eje entre los nodos i y j, y x_{ij} es 1 si y sólo si se utiliza dicho eje. Los valores c_{ij} indican el costo de utilizar el eje entre los nodos i y j.

De esta forma, la función objetivo para este modelo queda definida como:

$$Minimizar : \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.1}$$

Es decir, por cada eje, se suma su costo si este eje fue utilizado. Lo que se busca es minimizar la suma de los costos utilizados, sujeto a que los ejes elegidos cumplan con las restricciones necesarias para generar un circuito hamiltoniano, que es un ciclo que visita todos los puntos involucrados una única vez.

Para que este modelo represente satisfactoriamente al Problema del Viajante de Comercio, se utilizan las siguientes restricciones para modelar el hecho de que se está buscando un circuito hamiltoniano.

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n \tag{3.3}$$

$$0 \leq x_{ij} \leq 1, \qquad i, j = 1, \ldots, n \tag{3.4}$$

$$x_{ij} = 0, 1, \qquad i, j = 1, \ldots, n \tag{3.5}$$

La ecuación 3.2 indica que para todo nodo hay exactamente un eje de salida. Análogamente, la ecuación 3.3 indica que a cada nodo solamente se entra con un eje. Por último, 3.5 muestra el hecho que un eje entre dos nodos puede usarse o no, pero no se puede tomar un valor real intermedio.

Todas estas restricciones intentan modelar las condiciones necesarias para que la solución sea un camino hamiltoniano. Sin embargo, a este conjunto básico de desigualdades, que será reutilizado en varios modelos, le falta poder restringir el siguiente hecho:

Supongase que se tiene un mapa con 6 ciudades, lo que se quiere es encontrar un sólo circuito que pase por todas las ciudades una única vez. Sin embargo, con las restricciones actuales, la siguiente configuración es factible

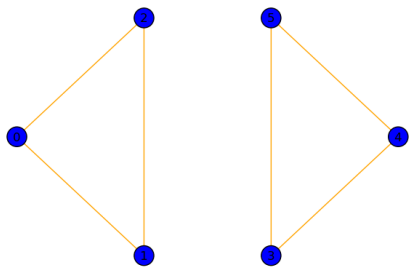

Figura 3.1: Solución con subtours

La situación planteada en la imagen anterior conforma un problema ya que no se quiere aceptar dichas soluciones. Se dice que la solución planteada posee *subtours*, y hasta este momento estas soluciones son factibles, ya que se cumplen todas las restricciones mencionadas.

Luego, la solución a este problema en el modelo es introducir nuevas restricciones que no permitan la formación de estos *subtours*. Estas restricciones suelen ser determinantes para los modelos de TSP, ya que son las que distinguen a los modelos entre sí, dado que casi todos los modelos conocidos en la literatura comparten las restricciones de asignación expuestas anteriormente.

Las restricciones que solucionan el hecho explicado, suelen denominarse *rompimiento de subtours*. En el caso de Dantzig, Fulkerson y Johnson, esta restricción queda plasmada como:

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad S \subseteq \{2, \ldots, n\}, \quad 2 \leq |S| \leq n - 1 \tag{3.6}$$

Todas las restricciones expuestas anteriormente, conforman el modelo original de Dantzig, Fulkerson y Johnson para el TSP. Este modelo tiene la particularidad de poseer una cantidad exponencial de restricciones al haber introducido los rompientos de subtours.

3.1.2. Modelo de Sherali y Driscoll(SD)

El modelo de Sherali y Driscoll, utiliza las mismas variables x_{ij} y la misma función objetivo que el modelo DFJ. Este modelo es una reformulación del modelo de Miller, Tucker y Zemlin(MTZ). Dicho modelo usa las variables u_i para definir el orden en el que el vértice i es visitado en el tour. De esta forma, además de utilizar las restricciones de asignación para los vértices, es decir las restricciones 3.2 y 3.3, utiliza las siguientes restricciones para modelar el rompimiento de subtours en base a las variables u recientemente mencionadas.

$$u_i - u_j + (n-1)x_{ij} \leq n - 2 \qquad i, j = 2, \ldots, n \tag{3.7}$$

$$1 \leq u_i \leq n - 1, \qquad i = 2, \ldots, n \tag{3.8}$$

En la formulación original MTZ, no se presentó 3.8, ya que con la primer restricción de las dos alcanza para romper los subtours.

Esto sucede porque si hubiese subtours, tiene que haber alguno que no incluya al vértice 1. En dicho subtour, la restricción 3.7 indica que si se va de i a j entonces el orden de i es menor, o analogamente $u_i \leq u_j$. Como este subtour no contiene al vértice 1, esta restricción se tiene que cumplir para todos los ejes. Sin embargo, eso generaría una relación de orden en todo el subtour que no puede existir dado que es un ciclo. Es por esto que la utilización de 3.7 impide una solución con subtours.

Luego, el modelo de SD es una reformulación del modelo MTZ, en donde se utilizaron técnicas de *lifteo* de restricciones y de *Reformulation-Linearization* para obtener restricciones más ajustadas. En este modelo, aparte de las variables x_{ij} e u_i, se utilizan las variables z_{ij} que representan el orden del eje ij en el tour.

Finalmente, el modelo queda definido por la función objetivo 3.1, las restricciones 3.2, 3.3 y:

$$\sum_{j=2}^{n} z_{ij} + (n-1)x_{i1} = u_i, \qquad i = 2, \ldots, n \tag{3.9}$$

$$\sum_{i=2}^{n} z_{ij} + 1 = u_j, \qquad j = 2, \ldots, n \tag{3.10}$$

$$x_{ij} \leq z_{ij} \leq (n-2)x_{ij}, \qquad i, j = 2, \ldots, n \tag{3.11}$$

$$u_j + (n-2)x_{ij} - (n-1)(1 - x_{ji}) \leq z_{ij} + z_{ji} \leq u_j - (1 - x_{ji}), \quad i, j = 2, \ldots, n \tag{3.12}$$

$$1 + (1 - x_{1j}) + (n-3)x_{j1} \leq u_j \leq (n-1) - (n-3)x_{1j} - (1 - x_{j1}), \quad j = 2, \ldots, n \tag{3.13}$$

3.1.3. Modelo Sarin, Sherali y Bhootra(SSB)

Este modelo propone una cantidad cúbica de restricciones para resolver el problema de los subtours. Este utiliza las mismas variables x_{ij} y la misma función objetivo que la primera formulación. Además utiliza las variables d_{ij}, estas variables binarias toman el valor 1 si y sólo si el vértice i precede al vértice j en el tour, sin la necesidad de ser el inmediato predecesor.

Con estas variables de precedencia, el modelo se conforma utilizando nuevamente las restricciones 3.2 y 3.3, para garantizar que a cada nodo se entra una vez y se sale una vez, junto con las siguientes restricciones:

$$d_{ij} \geq x_{ij} \qquad i, j = 2, \ldots, n \tag{3.14}$$

$$d_{ij} + d_{ji} = 1 \qquad i, j = 2, \ldots, n \tag{3.15}$$

$$d_{ij} + d_{jk} + d_{ki} \leq 2 \qquad i, j, k = 2, \ldots, n \tag{3.16}$$

La restricción 3.14 indica que si se usa el eje que va del vértice i al vértice j, entonces la variable d_{ij} debe plasmar el hecho de que i precede a j en el tour.

La ecuación 3.15 indica que si se toman dos vértices cualesquiera, sí o sí uno tiene que preceder al otro, y no lo pueden hacer mutuamente.

Por último, la restricción 3.16 se encuentra para restringir la existencia de un tour que no incluya al vértice 1. Esta restricción dice que si se toman 3 vértices, no puede haber un ciclo de precedencia entre ellos, rompiendo así la formación de cualquier subtour que no incluya al primer vértice.

Esta formulación, es la que en la literatura [18] se denomina SSB1.

Existen otras dos formulaciones más fuertes que provienen de agregar la restricción que indica que un nodo a lo sumo puede estar exactamente antes del principio del ciclo, o exactamente despues, pero no ambas:

$$x_{1j} + x_{j1} \leq 1 \qquad j = 2, \ldots, n \tag{3.17}$$

y de reemplazar la restricción 3.16 por la siguiente para dar lugar a la formulación SSB2

$$(d_{ij} + x_{ji}) + d_{jk} + d_{ki} \leq 2 \qquad i, j, k = 2, \ldots, n \tag{3.18}$$

o por esta otra opción para dar lugar a la formulación SSB3

$$3(d_{ij} + d_{jk} + d_{ki}) + x_{ji} + x_{kj} + x_{ik} \leq 6 \qquad i, j, k = 2, \ldots, n \tag{3.19}$$

Ambas desigualdades provienen de técnicas aplicadas a 3.16 para obtener una desigualdad más restrictiva.

3.1.4. Modelo de Sherali, Sarin y Tsai(SST)

En este contexto sólo se presentará el modelo SST2 que proviene de reformulaciones de los modelos SSB presentados anteriormente mezclado con anteriores modelos de los 3 autores.

El modelo SST2 consiste de la función objetivo 3.1 y las siguientes restricciones:

$$d_{ij} + d_{ji} = 1 \qquad i, j = 2, \ldots, n \tag{3.20}$$

$$(d_{ij} + x_{ji}) + d_{jk} + d_{ki} \leq 2 \qquad i, j, k = 2, \ldots, n \tag{3.21}$$

$$d_{ij} \geq x_{1i} \qquad i, j = 2, \ldots, n \tag{3.22}$$

$$d_{ji} \geq x_{i1} \qquad i,j = 2,\ldots,n \tag{3.23}$$

$$0 \leq t_{ij}^k \leq x_{ik} \qquad i,j,k = 2,\ldots,n \; i,k \neq j \tag{3.24}$$

$$\sum_{k=2;k\neq j}^{n} t_{ij}^k + x_{ij} = d_{ij} \quad i,j = 2,\ldots,n \tag{3.25}$$

$$x_{1k} + \sum_{i=2;i\neq j}^{n} t_{ij}^k = d_{kj} \quad k,j = 2,\ldots,n \tag{3.26}$$

Este modelo posee las mismas variables d_{ij} presentadas en SSB, y las variables de eje x_{ij}. Aparte posee unas nuevas variables t_{ij}^k que capturan la noción de ir desde el nodo i hasta el nodo j, siendo k el primer nodo intermedio entre ellos.

Con estas variables, el modelo hereda varias de las restricciones presentadas en SSB y presenta nuevas restricciones que tienen a las variables t_{ij}^k como protagonistas.

La restricción 3.24 indica que si se va de i a j pasando primero por $k(t_{ij}^k)$ es porque entonces se está usando el eje que va de i a k.

La restricción 3.25 dice que si i precede a j, es porque está inmediatamente antes(x_{ij}), o por que existe exactamente un k, distinto del nodo desde donde se comienza, a donde i se dirige primero para luego ir a j.

La restricción 3.26 indica que si k precede a j, entonces k es el segundo nodo del ciclo, o bien existe algún i para el cual k es el primer nodo en el camino entre i y j.

3.1.5. Modelo de Gavish y Graves(GG)

El modelo para el TSP de Gavish y Graves se enmarca en los denominados *Single Comodity Flow*. Lo que se busca en estos tipos de modelos es escribir las restricciones para el rompimiento de subtours en base a un flujo desde el vértice 1 a todos los demás vértices.

El modelo GG utiliza nuevamente las variables x_{ij} para indicar que ejes se utilizan en el tour. Este modelo se basa en la función objetivo 3.1 y las restricciones 3.2 y 3.3. Luego, para el rompimiento de subtours, el modelo cuenta con variables continuas no negativas g_{ij} que simulan un *flujo* presente en el arco ij. Con estas variables adicionales, se agregan las siguientes restricciones:

$$\sum_{j=1}^{n} g_{ji} - \sum_{j=2}^{n} g_{ij} = 1 \quad i = 2,\ldots,n \tag{3.27}$$

$$0 \leq g_{ij} \leq (n-1)x_{ij} \quad i = 1,\ldots,n;\; j = 2,\ldots,n \tag{3.28}$$

La idea del modelo es que el nodo 1 es el único capaz de *generar* flujo y enviarlo al resto de los nodos. Cada nodo en la solución va a tener que consumir unidades de ese flujo generado por el nodo 1.

La segunda restricción indica que para simular lo dicho anteriormente alcanza con usar flujos que vayan entre 0 y $n-1$, y asegura que si el eje no se va a utilizar en la solución entonces no se pase flujo por ese eje.

La primera ecuación está restrigiendo cuánto flujo tiene que consumir cada nodo, esta dice que la suma de los flujos que entran al nodo, menos la suma de los flujos que salen tiene que ser exactamente uno para todos los nodos que no son el distinguido por donde se comienza.

Es simple ver que estas restricciones evitan el problema de subtours, dado que si se presenta una solución con subtours, seguro tiene que existir un subtour donde no esté presente el nodo distinguido que puede generar flujo, entonces si empezamos a recorrer desde cualquier lado este subtour, se tiene que cumplir que en cada eje haya exactamente uno más de flujo que en el eje

siguiente, pero esto generaría una relación de orden imposible de existir en un ciclo. La idea es muy similar a la explicada anteriormente en el modelo SD.

3.1.6. Modelo de Fox, Gavish y Graves(FGG)

El modelo para TSP de Fox, Gavish y Graves tiene su origen en un problema de *machine scheduling* en donde una sola máquina tiene que procesar n trabajos y existe un costo por procesar un trabajo justo después de otro en determinado momento.

Esta formulación no utiliza las variables x_{ij}, sino que utiliza algo así como una descomposición de éstas. La formulación usa las variables binarias r_{ijk}, que indican si el vértice j se visita inmediatamente luego del i, en la posición k del tour. Dadas estas variables parecidas a las x_{ij} pero que distinguen la posición en el tour donde se usa el eje, se formula el siguiente modelo para el TSP.

$$Minimizar: \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk} r_{ijk} \tag{3.29}$$

Donde c_{ijk} es el costo de usar el eje ij en la posición k. En el caso del AngTSP usar el eje ij tiene el mismo costo sin importar en que momento del tour se lo use, por lo que los costos c_{ijk} tendrán el mismo valor para todo k, dados i y j.

Con esta función objetivo, las restricciones del modelo son

$$\sum_{j=1}^{n}\sum_{k=1}^{n} r_{ijk} = 1 \qquad i = 1 \dots n \tag{3.30}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} r_{ijk} = 1 \qquad j = 1 \dots n \tag{3.31}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} r_{ijk} = 1 \qquad k = 1 \dots n \tag{3.32}$$

$$\sum_{j=1}^{n}\sum_{k=2}^{n} k r_{ijk} - \sum_{j=1}^{n}\sum_{k=1}^{n} k r_{jik} = 1 \qquad i = 2 \dots n \tag{3.33}$$

Las primeras 3 restricciones son las análogas a las de asignación de los anteriores modelos, y la última restricción es la que restringe la creación de tours que no contengan al nodo distinguido 1, al crear una relación de orden entre los ejes análoga a la presentada en modelos anteriores.

Esta formulación presentada es la que en la literatura [18] se denomina FGG4. Dado que la tercera restricción, la 3.32, se puede sacar y el modelo sigue siendo correcto, se cuenta con la formulación FGG3 que consta simplemente de realizar esta eliminación.

3.1.7. Adaptación al AngTSP

En el caso del AngTSP, no sólo se quiere minimizar las distancias recorridas, sino que también se tienen en cuenta los ángulos de giro al realizar el tour.

Para extender la mayoría de los modelos que se presentan en esta sección, se utilizaron las variables binarias y_{ijk} que indican si en el tour se toma el ángulo de giro correspondiente a comenzar en el nodo i y pasar por el nodo j para ir al nodo k. Es claro que estas variables de ángulo tiene una correlación directa con las variables x_{ij} mencionadas anteriormente. En los casos de los modelos FGG, las variables x_{ij} no existen, pero son análogas a tomar las variables r y, dados i y j, sumar sobre todos los k posibles.

Dada la existencia de estas nuevas variables de ángulo, la función objetivo es modificada para tener en cuenta el hecho de que se quiere hacer un balance entre las distancias recorridas y los ángulos girados.

La nueva función objetivo queda modelada, para todos los modelos que tienen variables x_{ij}, como:

$$Minimizar : \alpha \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} + \beta \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \delta_{ijk} y_{ijk} \tag{3.34}$$

en donde los valores δ_{ijk} provienen de medir el ángulo que existe entre los nodos i,j y k. Los valores α y β son para pesar el balance entre minimizar la distancia y minimizar los ángulos. En el caso de tomar α como 1 y β como 0, entonces este modelo servirá para resolver el TSP.

En los modelos FGG, la función objetivo queda modelada de la siguiente manera:

$$Minimizar : \alpha \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} r_{ijk} + \beta \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \delta_{ijk} y_{ijk} \tag{3.35}$$

Luego, lo único restante para finalizar con la extensión al AngTSP, es agregar una restricción para definir la relación que hay entre las variables x y las variables y. Dicha restricción se modela como:

$$y_{ijk} \geq x_{ij} + x_{jk} - 1, \qquad i, j, k = 1, \ldots, n \tag{3.36}$$

Esta restricción fuerza el hecho de que si se usa el eje entre i y j, y también se usa el eje entre j y k, entonces la variable correspondiente al ángulo ijk también estará utilizada.

La restricción no es lo suficiente fuerte en el sentido de que se podría poner en 1 la variable y_{ijk}, sin necesidad de que los ejes ij y jk, hayan sido utilizados. Pero dado que la función objetivo es una minimización con coeficientes mayores o iguales a 0, siempre que se pueda tener las variables y_{ijk} en 0, la solución tomará esta opción.

En el caso de los modelos FGG, la restricción agregada es parecida, emulando las x con las sumatorias de las variables r sobre todos los k posibles. Quedando,

$$y_{ijk} \geq \sum_{t=1}^{n} r_{ijt} + \sum_{t=1}^{n} r_{tjk} - 1, \qquad i, j, k = 1, \ldots, n \tag{3.37}$$

Escalamiento de ángulos Algo importante a notar antes de comenzar con el estudio de las formulaciones es un hecho que se observó al realizar pruebas previas en los modelos para testear su corrección. Lo que se pudo observar es que los ciclos entregados como solución por cada modelo eran iguales para todas las combinaciones de alfa y beta que se probaban, excepto cuando el problema era puramente angular. Sumado al hecho de que era el mismo ciclo, se observaba que el valor de la función objetivo era siempre el valor que correspondía al caso del TSP, multiplicado por el alfa correspondiente al caso que se estuviese ejecutando.

Esto sucede porque así como está planteada la función objetivo, en la mayoría de las instancias comparar las distancias entre los puntos contra los ángulos entre los puntos no tiene ningún sentido práctico.

Las distancias pueden tomar cualquier valor, dependiendo que tan lejos estén los puntos en una determinada instancia; mientras que los ángulos, sin importar la instancia, no pueden valer más de Pi radianes.

Luego, si bien el beta intenta darle peso a los ángulos, lo más conveniente sigue siendo tomar el mejor circuito en cuanto a distancias para minimizar la función objetivo total; exceptuando el caso puramente angular en donde las distancias ni siquiera son tenidas en cuenta.

Para solucionar este problema de inconmesurabilidad entre los ángulos y las distancias se optó por escalar los ángulos de manera tal que los valores que tomen sean comparables a las distancias. Para hacer esto de una forma tal que los ángulos no pierdan la relación entre ellos, lo que se hace en cada instancia es multiplicar cada ángulo por la distancia máxima presente en esa instancia, dividido Pi. De esta forma los ángulos no pierden la proporción entre ellos al multiplicarse por una constante, y se logra que ahora se distribuyan en el mismo intervalo que las distancias.

Se consideraron otras opciones posibles para solucionar este problema. Una de ellas era escalar tanto el alfa como el beta, pero se descartó esta posibilidad para poder mantener el hecho de que estos dos parámetros sumen 1 para los casos de prueba.

Por otro lado, estaba la posibilidad de escalar las distancias en vez de los ángulos, pero se desestimó esta posibilidad porque el hecho de dejar fijas las distancias hace que no se pierda el chequeo de corrección de la resolución de los casos con alfa igual a 1 contra benchmarks reconocidos para el TSP.

3.2. Comparación de formulaciones

Para poder continuar el análisis más profundo del AngTSP, se realizaron pruebas preliminares sobre los modelos anteriormente explicados con el fin de elegir el modelo que resulte más adecuado para su posterior fortalecimiento.

Lo que se analizó fue la implementación más simple de las formulaciones en el framework CPLEX. Es decir, se implementaron las formulaciones explicitando las variables, la función objetivo y las restricciones, y se dejó en manos del paquete CPLEX todo el manejo de un branch-and-cut por defecto. Es importante mencionar que, dado que el modelo DFJ tiene una cantidad exponencial de restricciones, estas se tratan de forma dinámica, agregandose al modelo a medida que va siendo necesario.

Todos los análisis hechos en esta sección, como así también en las etapas posteriores del trabajo, fueron realizados sobre las mismas máquinas y las mismas instancias.

Las máquinas utilizadas para la ejecución tienen procesadores Intel i7 de 3.40GHz y ejecutan CPLEX 12.3 con 16Gb de RAM disponibles.

Las instancias que se utilizaron para el análisis son seis instancias generadas aleatoriamente eligiendo la cantidad de puntos entre 1 y 50 con una distribución uniforme. Asimismo, para la posición de los puntos se eligieron coordenadas en el plano aleatoriamente con distribución uniforme entre el 0 y el 2000. Luego, cada instancia se probó con 5 pares de *alfas* y *betas*. Se eligió que ambos parámetros sumasen 1 en todos los casos y los *alfas* elegidos fueron 1.0(TSP), 0.75, 0.50, 0.25 y 0.0(puramente angular).

Las instancias utilizadas tienen 16, 35, 25, 12, 31 y 46 nodos respectivamente

A continuación se presentan los resultados obtenidos para cada modelo, en cada uno de los aspectos que se consideraron relevantes. Las ejecuciones de los branch-and-cuts se realizaron con un tiempo límite de media hora para cada instancia y combinación de parámetros.

3.2.1. Tiempos de ejecución

En esta sección, se presentan los tiempos de ejecución de cada modelo sobre cada uno de los pares instancia-parámetros mencionados anteriormente. Todos los tiempos presentados a lo largo de los diferentes análisis serán en segundos.

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.08	8.81	399.10	1800	1800
SD	0.18	12.64	1574.13	1800	1800
GG	0.13	14.58	1104.07	1800	1800
SSB1	0.29	15.80	1800	1800	1800
SSB2	0.56	31.50	1800	1800	1800
SSB3	0.41	25.72	1800	1800	1800
SST2	1.46	691.84	1800	1800	1800
FGG3	25.71	1800	1800	1800	1800
FGG4	41.28	1048.87	1800	1800	1800

Cuadro 3.1: Tiempos para Instancia Random 1

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	3.57	1800	1800	1800	1800
SD	41.79	1800	1800	1800	1800
GG	0.89	1800	1800	1800	1800
SSB1	839.57	1800	1800	1800	1800
SSB2	1740.10	1800	1800	1800	1800
SSB3	329.10	1800	1800	1800	1800
SST2	465.11	1800	1800	1800	1800
FGG3	1800	1800	1800	1800	1800
FGG4	1800	1800	1800	1800	1800

Cuadro 3.2: Tiempos para Instancia Random 2

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.33	1800	1800	1800	1800
SD	0.17	1800	1800	1800	1800
GG	0.14	1800	1800	1800	1800
SSB1	0.87	1800	1800	1800	1800
SSB2	3.05	1800	1800	1800	1800
SSB3	0.58	1800	1800	1800	1800
SST2	19.26	1800	1800	1800	1800
FGG3	1800	1800	1800	1800	1800
FGG4	844.87	1800	1800	1800	1800

Cuadro 3.3: Tiempos para Instancia Random 3

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.04	2.80	36.57	420.71	378.16
SD	0.34	6.48	112.46	712.96	957.21
GG	0.12	6.24	37.70	448.35	535.97
SSB1	0.07	4.99	54.06	513.21	1627.12
SSB2	0.50	8.30	216.95	1345.46	1800
SSB3	0.11	4.53	66.52	916.09	1800
SST2	0.12	65.81	1800	1800	1800
FGG3	1.28	150.99	1800	1800	1800
FGG4	1.52	77.70	1800	1800	1800

Cuadro 3.4: Tiempos para Instancia Random 4

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	2.32	1800	1800	1800	1800
SD	127.86	1800	1800	1800	1800
GG	0.84	1800	1800	1800	1800
SSB1	141.49	1800	1800	1800	1800
SSB2	189.04	1800	1800	1800	1800
SSB3	155.80	1800	1800	1800	1800
SST2	491.63	1800	1800	1800	1800
FGG3	1800	1800	1800	1800	1800
FGG4	1800	1800	1800	1800	1800

Cuadro 3.5: Tiempos para Instancia Random 5

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	12.95	1800	1800	1800	1800
SD	119.72	1800	1800	1800	1800
GG	17.59	1800	1800	1800	1800
SSB1	1800	1800	1800	1800	1800
SSB2	1800	1800	1800	1800	1800
SSB3	1800	1800	1800	1800	1800
SST2	1800	1800	1800	1800	1800
FGG3	1800	1800	1800	1800	1800
FGG4	1800	1800	1800	1800	1800

Cuadro 3.6: Tiempos para Instancia Random 6

En primer lugar, se puede observar que la mayoría de las ejecuciones realizadas tomaron todo el tiempo posible para intentar resolver el problema. La diferencia que existe entre los casos donde el problema es puramente el TSP y cuando las variables de ángulo entran en juego en la función objetivo es notoria. Esto sucede gracias a las herramientas de análisis que tiene el framework CPLEX.

En el caso del TSP, CPLEX identifica que las variables de ángulo no son importantes para

la función objetivo y trata de fijarlas en algún valor para que se cumplan todas las restricciones pertinentes. En este caso, al no estar las variables de ángulo en la función objetivo, se las puede fijar en uno a todas ellas y, de esta manera, se estarán cumpliendo todas las restricciones de relación entre las variables x y las variables y, sin afectar a la función objetivo. Por lo que al hacer esta fijación el problema se reduce a uno donde se tiene una cantidad cuadrática de variables, en vez de la cantidad cúbica de variables originales, y muchas menos restricciones para cumplir, ya que se elimina una cantidad cúbica de desigualdades.

Cuando la función objetivo empieza a depender de las varibles de ángulo, ya no se puede hacer esta fijación y descarte de variables y restricciones, lo que deviene en un problema mucho más complejo de resolver.

Aunque la mayoría de las instancias no fueron resueltas en el tiempo fijado, sí hay una cierta cantidad que fueron resueltas por varios de los modelos. En estos casos, se puede ver como la formulación de Dantzig, Fulkerson y Johnson es la que se destaca, siendo solamente comparable con la formulación de Gavish y Graves, que de todas maneras sólo logra superar a la DFJ en tres ejecuciones.

Se puede observar que todas las demás formulaciones logran tiempos de resolución considerablemente más altos para las pocas instancias resueltas.

Si bien estas evidencias muestran una tendencia, es importante continuar con el análisis de otros aspectos, ya que la cantidad de instancias que fueron resueltas en el tiempo máximo fijado fueron pocas en relación a las que se intentó resolver. Es importante analizar cómo fue el desempeño de las diferentes formulaciones en las instancias que no se pudieron resolver, por ejemplo analizando las cotas primales y duales alcanzadas.

3.2.2. Cotas

En esta sección se presentan las mejores cotas inferiores y superiores encontradas por cada formulación. Las cotas primales son la mejor solución encontrada hasta el momento por el algoritmo, mientras que las cotas duales son, en el caso de una minimización como esta, el valor más alto del cual se tiene certeza de que la función objetivo no puede ser menor a él. Dadas las anteriores definiciones, se entiende que cuando la cota primal coincide con la dual el problema fue resuelto hasta la optimalidad ya que se tiene una solución factible, y también se tiene la certeza de que no se puede mejorar.

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	6874 6874	7693.13 7693.13	8080.41 8080.41	9069.42 5755.30	9403.65 3599.40
SD	6874 6874	7693.13 7693.13	8080.41 8080.41	10477.1 4953.83	10558.2 2224.33
GG	6874 6874	7693.13 7693.13	8080.41 8080.41	9108.85 5465.02	9384.82 3060.66
SSB1	6874 6874	7693.13 7693.13	8080.41 7861.26	10168.8 4969.13	10689.8 1883.12
SSB2	6874 6874	7693.13 7693.13	9303.89 6824.70	9158.39 4403.12	12319.0 581.27
SSB3	6874 6874	7693.13 7693.13	8536.32 7171.10	11054.1 4508.55	12150.0 594.35
SST2	6874 6874	7693.13 7693.13	9831.82 5450.04	10406.9 2842.84	14954.8 0.0000
FGG3	6874 6874	7966.52 6953.58	10493.2 5612.71	12252.3 3012.25	9927.32 0.0000
FGG4	6874 6874	7693.13 7693.13	10800.3 6145.98	9375.56 3245.09	11506.4 0.0000

Cuadro 3.7: Cotas Instancia Random 1

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	10407 10407	16057.38 11346.81	21813.62 8419.10	26133.22 4329.49	34599.33 0.00
SD	10407 10407	17933.35 11535.96	22332.14 8703.98	23707.04 4296.36	32163.14 0.00
GG	10407 10407	18487.68 11463.34	21573.34 8245.03	23589.08 4358.22	31410.22 0.00
SSB1	10407 10407	19333.35 10997.01	26969.96 7638.56	34477.52 3881.00	62291.80 0.00
SSB2	10407 10407	19309.62 11154.11	25677.69 7740.99	33162.94 3932.57	53565.19 0.00
SSB3	10407 10407	18079.64 11251.96	25469.64 7799.03	33712.23 3905.10	49603.90 0.00
SST2	10407 10407	17884.58 8650.96	23283.16 5763.95	28681.74 2876.36	INF 0.00
FGG3	INF 9401.3	INF 8279.67	INF 5957.03	INF 3167.88	33357.64 0.00
FGG4	14219 9396.4	25590.17 9068.07	INF 6210.91	39276.37 3087.80	45077.72 0.00

Cuadro 3.8: Cotas Instancia Random 2

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	7754 7754	9823.29 9428.37	11607.81 7665.86	14487.30 4088.56	14024.87 488.82
SD	7754 7754	9859.35 9337.63	13434.81 7227.22	14269.21 4081.90	18767.32 0.00
GG	7754 7754	9823.29 9276.72	12876.10 7067.55	16496.34 4354.72	17380.43 429.91
SSB1	7754 7754	10304.33 8946.78	14580.47 7184.87	18198.67 3771.43	26937.41 0.00
SSB2	7754 7754	10974.43 8548.22	15398.73 6604.44	18208.07 3562.01	25259.18 0.00
SSB3	7754 7754	10126.46 8874.38	14673.15 6571.18	16232.13 3396.23	22464.84 0.00
SST2	7754 7754	11786.92 8151.99	14911.10 5988.77	15856.21 3002.95	32619.41 0.00
FGG3	INF 7180.83	INF 8294.18	INF 5967.72	INF 3276.75	18545.17 0.00
FGG4	7754 7754	11825.78 8244.89	14743.36 5939.21	14619.83 3189.00	20995.32 0.00

Cuadro 3.9: Cotas Instancia Random 3

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	8716.38 8716.38
SD	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	8716.38 8716.38
GG	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	8716.38 8716.38
SSB1	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	8716.38 8716.38
SSB2	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	9231.41 1755.02
SSB3	6311 6311	7306.80 7306.80	7868.69 7868.69	8292.53 8292.53	8716.38 7054.84
SST2	6311 6311	7306.80 7306.80	7868.69 6552.10	9240.26 4722.46	10186.8 0.00
FGG3	6311 6311	7306.80 7306.80	8009.99 6703.61	8643.70 4766.51	10686.8 1580.37
FGG4	6311 6311	7306.80 7251.42	8132.13 6693.07	8600.08 4945.41	9716.17 1847.40

Cuadro 3.10: Cotas Instancia Random 4

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	9447 9447	14543.46 10814.14	20112.56 8123.57	22452.62 4259.18	28888.70 0.00
SD	9447 9447	15493.77 10781.67	19088.35 8109.73	22459.59 4087.68	47472.95 0.00
GG	9447 9447	15332.97 10729.90	19234.79 8325.20	23274.14 4323.58	26675.85 0.00
SSB1	9447 9447	16966.46 10493.27	22146.26 7466.02	26430.58 3817.99	51598.10 0.00
SSB2	9447 9447	16747.16 10248.87	23342.05 7667.44	28785.89 3745.42	40614.84 0.00
SSB3	9447 9447	17295.10 10280.07	22067.93 7579.54	26478.59 4063.26	37140.38 0.00
SST2	9447 9447	16002.90 8655.89	21426.80 6057.67	26850.70 3064.77	47223.29 0.00
FGG3	10681 9324.50	INF 8321.72	INF 5263.65	INF 2683.31	24275.98 0.00
FGG4	9560 8875.68	19679.13 9090.20	24933.39 5609.39	INF 2907.56	INF 0.00

Cuadro 3.11: Cotas Instancia Random 5

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	11168.00 11168.00	19171.21 12243.98	26550.31 9045.86	34047.95 4502.22	47160.55 0.00
SD	11168.00 11168.00	21630.50 12318.95	27954.58 8771.90	58897.45 4601.58	70569.16 0.00
GG	11168.00 11168.00	21846.19 12022.00	45355.25 8603.80	41923.08 4572.48	35310.13 0.00
SSB1	11168.00 11127.60	25746.38 12075.06	INF 8564.78	4402.35 43130.61	70479.10 0.00
SSB2	12275.00 10908.07	27569.88 10812.03	35771.01 7270.10	47187.69 3652.24	7062.93 0.00
SSB3	13634.00 10887.00	22198.67 11571.53	32728.68 7740.72	42870.47 3943.09	72327.99 0.00
SST2	INF -INF	INF -INF	INF -INF	INF -INF	INF -INF
FGG3	INF 10255.97	INF 8968.09	INF 7077.20	INF 3466.08	INF 0.00
FGG4	18772.00 10021.41	INF 10632.17	INF 6938.50	INF 3674.84	INF 0.00

Cuadro 3.12: Cotas Instancia Random 6

En primer lugar se nota que, como era de esperar, en la mayoría de los casos se puede ver distinta cota inferior y superior, ya que como se vió en la sección anterior, muchas de las formulaciones tomaron todo el tiempo de ejecución posible en varias de las instancias. El análisis interesante en esta sección se dará justamente en esos casos para ver que cotas primales y duales se alcanzaron.

Observando las diferentes tablas, lo que se busca es obtener cotas duales elevadas y cotas primales bajas. Es decir, que el intervalo donde vive el valor óptimo sea el más chico posible. Si bien ambas cotas son importantes, para este análisis resultará de más importancia la cota inferior, ya que lo que se quiere analizar es que tan ajustadas son las relajaciones lineales de las formulaciones, y además porque una formulación con malas cotas primales puede ser fácilmente mejorada con la presencia de heurísticas para crear soluciones factibles de calidad.

Analizando las cotas duales, en la mayoría de las ejecuciones hechas se puede segmentar las formulaciones en tres grupos, aunque haya instancias donde algunos resultados se entremezclen.

En un primer grupo se ubican las formulaciones DFJ, GG y SD, consiguiendo las mejores cotas duales en absolutamente todos los casos presentados. Luego, en un segundo grupo, se pueden encontrar las formulaciones SSB1, SSB2 y SSB3, que en la mayoría de los casos obtienen cotas duales de menor calidad que el grupo anterior, y también suelen presentar una cota primal más elevada. Por último, se encuentran las formulaciones SST2, FGG3 y FGG4 que poseen las peores cotas duales, incluso con casos en donde no se encuentran cotas duales ya que el tiempo fijado no alcanza para resolver la relajación lineal del nodo raíz.

En cuanto a los casos donde *alfa* es igual a 0, todos los modelos se encuentran con el siguiente problema.

La cota inferior inicial de los modelos está dada por la solución de la relajación lineal. Al ser *alfa* igual a 0, la parte de la función objetivo que habla sobre distancias no tiene injerencia en la solución final. Es por esto que la mejor solución de la relajación puede tomar cualquier valor conveniente en las variables x_{ij}, ya que esto no afectará al valor objetivo. Luego, lo que sucede con todos los modelos es que para resolver la relajación lineal conviene hacer que todas las x_{ij} valgan $1/N$, donde N es la cantidad total de puntos. Al tomar las x_{ij} de esta manera, se cumplen todas las restricciones que no incluyen a las variables de ángulo. Una vez asignadas las variables x_{ij} con este valor, se puede observar que a todas las variables y_{ijk} se les puede asignar un 0, ya que esto cumplirá satisfactoriamente con la restricción 3.36 y, a la vez, minimizará la función objetivo dándole también un valor de 0.

Si bien hay modelos que no poseen las variables x, sucede la situación análoga con las variables que utiliza.

Es por esto que todos los modelos comienzan con la peor cota inferior posible y luego cuesta mucho achicar el gap que se produce originalmente.

Al igual que sucedió con el análisis de tiempos de ejecución, nuevamente se puede ver que los modelos DFJ y GG son los que mejores se comportan. Si bien no hay un dominio absoluto, el modelo DFJ tiene mejor cota primal y dual que el modelo GG en más casos de los que se observa el orden inverso. Por lo cual, se continúa con la tendencia de posicionar a estas dos formulaciones por sobre las demás, y de quedarse con la formulación de Dantzig, Fulkerson y Johnson como la mejor de las analizadas.

3.2.3. Tiempos de las Relajaciones lineales

A continuación se presentan los tiempos que cada formulación tomó para resolver la relajación lineal del nodo raíz.

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.01	0.01	0.00	0.00	0.01
SD	0.01	0.02	0.03	0.02	0.04
GG	0.00	0.01	0.01	0.01	0.02
SSB1	0.00	0.01	0.01	0.01	0.03
SSB2	0.03	0.05	0.06	0.02	0.16
SSB3	0.01	0.01	0.01	0.01	0.06
SST2	1.08	1.14	1.21	1.22	2.59
FGG3	0.00	0.01	0.01	0.01	0.08
FGG4	0.01	0.06	0.07	0.06	0.13

Cuadro 3.13: Tiempo Relajación Instancia Random 1

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.04	0.04	0.04	0.04	0.15
SD	0.22	0.37	0.48	0.48	2.16
GG	0.02	0.16	0.16	0.16	0.32
SSB1	0.94	1.84	1.85	1.85	8.37
SSB2	1.42	3.90	4.19	5.43	38.75
SSB3	3.64	4.10	3.25	3.49	22.41
SST2	464.73	547.08	538.86	544.39	758.36
FGG3	0.04	0.41	0.40	0.41	2.82
FGG4	0.22	1.64	1.65	1.64	8.49

Cuadro 3.14: Tiempo Relajación Instancia Random 2

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.02	0.02	0.01	0.01	0.04
SD	0.04	0.17	0.10	0.10	0.37
GG	0.01	0.03	0.04	0.04	0.11
SSB1	0.02	0.04	0.04	0.04	0.62
SSB2	0.52	0.45	0.37	0.37	4.02
SSB3	0.33	0.18	0.22	0.26	2.15
SST2	19.14	21.77	22.29	18.97	63.72
FGG3	0.01	0.09	0.09	0.09	0.56
FGG4	0.06	0.40	0.53	0.53	1.02

Cuadro 3.15: Tiempo Relajación Instancia Random 3

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.00	0.00	0.00	0.00	0.00
SD	0.01	0.01	0.01	0.01	0.01
GG	0.00	0.00	0.00	0.00	0.01
SSB1	0.00	0.00	0.00	0.00	0.00
SSB2	0.00	0.01	0.01	0.01	0.01
SSB3	0.00	0.00	0.00	0.00	0.01
SST2	0.11	0.17	0.19	0.19	0.26
FGG3	0.00	0.00	0.00	0.00	0.02
FGG4	0.00	0.01	0.01	0.01	0.03

Cuadro 3.16: Tiempo Relajación Instancia Random 4

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.02	0.02	0.02	0.02	0.15
SD	0.12	0.22	0.21	0.22	1.09
GG	0.02	0.09	0.08	0.08	0.21
SSB1	0.55	0.07	0.07	0.07	3.64
SSB2	0.63	0.76	0.90	0.91	15.43
SSB3	0.63	0.79	0.66	0.66	10.72
SST2	126.98	116.39	127.93	126.09	343.45
FGG3	0.02	0.23	0.23	0.23	1.69
FGG4	0.14	1.30	1.30	1.31	3.34

Cuadro 3.17: Tiempo Relajación Instancia Random 5

Modelo \ Alfa	1.0	0.75	0.50	0.25	0.0
DFJ	0.09	0.10	0.10	0.10	0.67
SD	0.51	1.71	1.31	1.30	17.11
GG	0.09	0.49	0.48	0.49	0.66
SSB1	9.58	32.43	32.83	32.51	110.08
SSB2	36.75	37.39	38.34	38.23	756.76
SSB3	13.80	42.41	47.43	47.54	215.05
SST2	1800	1800	1800	1800	1800
FGG3	0.10	1.12	1.12	1.11	15.60
FGG4	1.60	13.20	15.52	12.66	34.22

Cuadro 3.18: Tiempo Relajación Instancia Random 6

Este apartado de los resultados sirve para ver qué tan rápido se resuelve una relajación lineal de cada formulación, ya que esto tendrá una correlación directa con la cantidad de nodos que se pueden llegar a explorar en un tiempo de ejecución fijo.

Nuevamente se puede observar la misma tendencia dada en los aspectos anteriores. Las formulaciones DFJ y GG son las que presentan relajaciones lineales más livianas, obteniendo tiempos de resolución bien distinguidos respectos de las otras formulaciones.

Lo único diferente respectos de los análisis anteriores, es que ahora las formulaciones FGG3 y FGG4 no son de las peores, sino que se encuentran justo detrás de las dos mejores formulaciones. Sin embargo, tanto en los tiempos de ejecución, como en las cotas obtenidas, quedan muy por detrás de las demás formulaciones como para ser tenidas en cuenta en la comparación global.

Dados todos los resultados presentados anteriormente, el análisis generado a partir de los mismos y la evidencia bibliográfica; se optó por continuar el trabajo con la formulación de Dantzig, Fulkerson y Johnson, para un posterior fortalecimiento.

En los próximos capítulos se presentarán los principales componentes del algoritmo Branch-and-Cut diseñado para este problema.

Durante el transcurso de estos se irán explayando dichos componentes, como por ejemplo las heurísticas iniciales utilizadas, las heurísticas primales, las decisiones de branching tomadas y las desigualdades válidas formuladas.

De cada uno de estos aspectos se irá realizando un estudio objetivo de las diferentes opciones para lograr determinar una buena combinación de los mismos.

Capítulo 4

Heurísticas Iniciales

Uno de los primeros aspectos a tener en cuenta en un algoritmo branch-and-cut es la utilización de una heurística inicial razonable.

Una heurística inicial es aquella que se utiliza una única vez al comenzar con un problema para obtener una primera solución factible que se encuentre tan cerca del óptimo como se pueda, con el objetivo de tener una cota superior inicial para comenzar el branch-and-cut.

Como casi todo aspecto práctico, la elección de una buena heurística inicial está sujeta a varios *tradeoffs*. Por un lado es muy útil desarrollar una buena heurística que entregue una solución factible cercana al óptimo. Esto resultará en *gaps* más pequeños entre las cotas primales y duales, lo que puede devenir en mejores tiempos de ejecución. Esta mejora está sustentada en el hecho de que encontrar una buena cota primal de manera temprana, puede ser beneficioso para podar grandes partes del árbol de branch-and-cut. Si en un nodo se encuentra una cota dual mayor a la primal obtenida, es seguro podar todo el subárbol que cuelga de este nodo ya que no se podrá encontrar mejor solución en este sector que la que ya se tiene.

Por otro lado, es importante no poner un esfuerzo computacional excesivo en la heurística inicial, que pueda perjudicar la performance del algoritmo en su totalidad.

A continuación se presentan las heurísticas iniciales analizadas para el Ang-TSP, junto con dos metaheurísticas simples de búsqueda local para mejorar las soluciones iniciales.

4.1. Ángulo Greedy

Esta primera heurística se pensó para los casos en los que los ángulos tienen más peso que las distancias en la función objetivo, es decir cuando el *beta*, peso correspondiente a las variables de ángulo, es mayor al *alfa*, peso correspondiente a las variables de distancia.

Esta heurística corresponde a la categoría de algoritmos golosos. Estos son algoritmos que en cada paso del mismo toman una decisión de cómo continuar en base a elegir la mejor opción dado un criterio provisto. Esta decisión se realiza sin considerar las implicaciones de la misma en los pasos futuros ni en la solución final, y sin tener la opción de poder revisarla *a posteriori*.

Dichos algoritmos son muy útiles para ciertos problemas, conformando soluciones exactas para ellos. En otros casos, como el que se está tratando, el algoritmo goloso no resuelve las instancias de manera exacta, pero produce una solución factible para el problema.

La heurística de *Ángulo Greedy* se basa en comenzar desde dos nodos arbitrarios y luego ir agregando en la solución uno a uno los nodos que todavía no fueron visitados. En cada paso el nodo no visitado que se agrega a la solución es el que genera un menor ángulo con los dos últimos nodos visitados.

De esta manera, la decisión golosa esta justamente basada en hacer tan chicos los ángulos como se pueda en cada momento.

A continuación se presenta el pseudocódigo de la heurística *Ángulo greedy*

```
function AnguloGreedyConDosNodos(primerNodo, segundoNodo)
    resultado ← []
    agregar primerNodo a la cola de resultado
    agregar segundoNodo a la cola de resultado
    while resultado no contenga todos los nodos do
        actual ← nodo no utilizado que forme mejor angulo con los dos ultimos de resultado
        agregar actual a la cola de resultado
    end while
    devolver resultado
end function
```

Como se puede observar en el anterior pseudocódigo, una vez que se tienen los dos nodos iniciales se tienen que ir agregando uno a uno los nodos que todavía no están visitados, por cada una de estas inserciones se pregunta cuál es el nodo entre todos los no visitados que generaría un menor ángulo.

De esta manera, al tener los dos nodos iniciales, la heurística tendría una complejidad temporal perteneciente a $\mathcal{O}(n^2)$, ya que se agregan al ciclo todos los nodos, y por cada inserción se recorren todos los nodos no visitados para saber cual es el candidato a insertar.

Sin embargo, es muy restrictivo elegir dos nodos arbitrarios para comenzar ya que esto significaría que el eje que los une está siempre presente en la solución, lo cual restringe mucho el espacio de soluciones posibles. Es por esto que se optó por fijar solamente el nodo inicial y luego correr la heurística con todas las distintas posibilidades para el segundo nodo. Así, la heurística pasa a tener una complejidad temporal pertenciente a $\mathcal{O}(n^3)$. Quedando la siguiente heurística

```
function AnguloGreedy(nodo1)
    mejorValor ← ∞
    mejorSolucion ← []
    for segundoNodoActual = 2 → cantidadNodos do
        solucionActual ← AnguloGreedyConDosNodos(nodo1, segundoNodoActual)
        if valorObjetivo(solucionActual) < mejorValor then
            mejorValor ← valorObjetivo(solucionActual)
            mejorSolucion ← solucionActual
        end if
    end for
    devolvermejorSolucion
end function
```

Si bien es cierto que fijar el primer nodo es restrictivo en cuanto al orden impuesto para correr la heurística, se decidió dejarlo así dado que variar también este primer nodo haría que la heurística tuviese una complejidad temporal perteneciente a $\mathcal{O}(n^4)$, lo cual representa mucho esfuerzo computacional para la cantidad de nodos que se quieren manejar con este branch-and-cut, sobre todo teniendo en cuenta que recién se están analizando heurísticas iniciales. Sí se consideró importante variar el segundo nodo fijado ya que este no solo imponía orden sino que, como ya se mencionó, imponía la utilización de un eje en la solución, lo cual es bastante más restrictivo.

A continuación se presenta la solución inicial entregada por la heurística para la instancia random número 1. Se presentan tanto la solución cuando el problema es el TSP, como cuando el problema es puramente angular. Para esta instancia, esta heurística arrojó el mismo resultado para ambas combinaciones de parámetros.

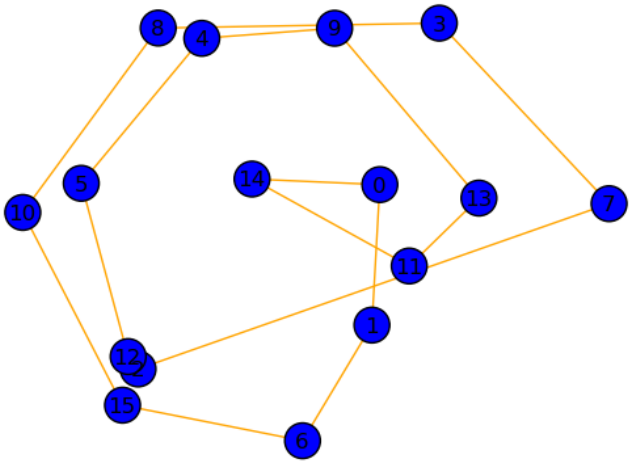

Figura 4.1: AnguloGreedy para instancia random1

Lo que se puede observar es cómo la heurística al buscar el menor ángulo de giro posible termina encontrando soluciones que se parecen a ir armando cápsulas convexas concéntricas de todos los puntos, exceptuando por los puntos cercanos al 0 donde el ciclo está obligado a empezar y terminar.

4.2. Farthest Insertion

En contraposición a la heurística de Ángulo greedy, en esta sección se presenta una heurística que está pensada para los casos en los que es más importante minimizar la distancia que se recorre y no los ángulos. *Farthest Insertion* es una heurística reconocida en la literatura [11] para el problema de viajante de comercio.

La heurística de *farthest insertion* se basa en ir formando un ciclo con todos los nodos, agregando en cada paso un nuevo nodo al ciclo ya existente.

Se comienza con un nodo cualquiera arbitrario, que conforma un ciclo consigo mismo, y en cada paso se aumenta en uno la cantidad de nodos que estan presentes en el ciclo resultado.

El procedimiento para agregar un nodo consta de dos fases, en una primera etapa se busca cuál es el nodo que todavía no fue visitado que se va a ingresar al ciclo. Para elegir este nodo lo que se hace es buscar el nodo no visitado que esté más lejos a todos los nodos que ya forman parte del ciclo. Esta elección es la que le da el nombre a la heurística y en la literatura consultada ha mostrado mejores desempeños que otras heurísticas del grupo como *Nearest Insertion*, en la cual se elige al nodo que esta más cerca a los ya presentes en el ciclo.

Una vez que se tiene el nodo que se va a insertar, se procede con la segunda fase de la inserción que consta de buscar el mejor el lugar para insertarlo, para esto se analiza cada posición posible en el ciclo para insertarlo y se elige golosamente la posición que minimice la función objetivo.

A continuación se presenta el pseudocódigo de la heurística

function FarthestInsertion
  resultado ← []
  agregar el nodo 1 *a resultado*
  while *resultado no tenga todos los nodos* **do**
    actual ← *nodo no usado con mayor distancia a todos los del ciclo*
    posicion ← *mejor posicion, segun la funcion objetivo, para insertar actual*
    agregar actual a resultado en posicion

end while
devolver resultado
end function

A continuación se presentan las soluciones encontradas por la heurística para la instancia random1, tanto para el problema TSP, como para el problema puramente angular, para visualizar gráficamente como se comporta la heurística en los dos casos extremos, y como se compara visualmente lo hecho por esta heurística con lo hecho por *ánguloGreedy*.

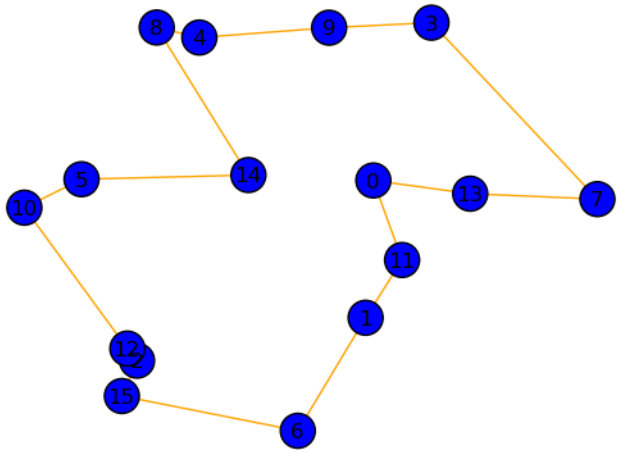

Figura 4.2: FarthestInsertion para instancia random1 TSP

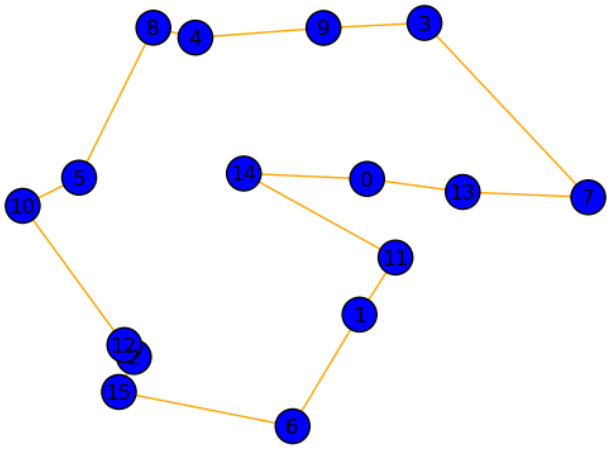

Figura 4.3: FarthestInsertion para instancia random1 angular

De los resultados anteriores se puede observar que la heurística se comporta lógicamente diferente dependiendo los parámetros que se estén utilizando. En el caso del problema puramente angular se presenta un ciclo con un solo cambio de dirección muy brusco entre los nodos 0, 14 y 11, como para que la suma de los ángulos sea lo más chica posible. Por otro lado, en el caso del

TSP se ven dos de estos giros bruscos, pero en pos de tener una mejor separación entre los nodos del lado izquierdo y los de lado derecho para minimizar la distancia total recorrida.

Como se puede observar, las soluciones arrojadas por esta heurística son sustancialmente diferentes a las entregadas para *ánguloGreedy*.

4.3. Búsqueda Local: 2-opt

La heurística de búsqueda local 2-opt es un método conocido en la literatura para poder reforzar los ciclos encontrados por las heurísticas constructivas presentadas anteriormente. La idea es definir una vecindad para un ciclo, conformada por todos los ciclos provenientes de tomar el original, borrar dos ejes no consecutivos y reconstruir el ciclo de la única forma posible. Como se ve en la siguiente figura, una vez que se toma un ciclo y se eligen los ejes, solo hay una forma posible de reconectar los nodos para que formen un ciclo.

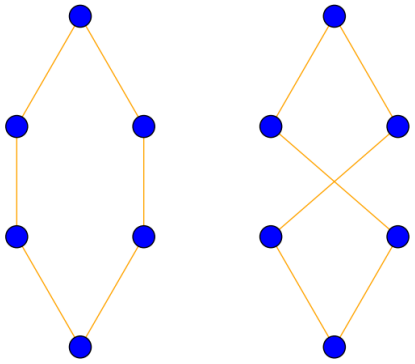

Figura 4.4: Única reconfiguración del ciclo

La búsqueda local 2-opt no construye un ciclo, sino que toma un ciclo ya factible y se mueve por la vecindad buscando un ciclo de menor valor objetivo. Luego de pasar a este nuevo ciclo, repite el proceso de buscar en la vecindad del mejor ciclo actual, hasta caer en un ciclo que sea el óptimo dentro de su vecindad. Es por esto que 2-opt no se utilizará como una heurística por si misma, sino que se usará para mejorar las soluciones que ya se obtuvieron por *ánguloGreedy* y *farthestInsertion*.

A continuación se presentan los ciclos entregados por 2-opt cuando toma como entrada el ciclo entregado por *ánguloGreedy*. Los casos mostrados son los mismos que en las heurísticas previas para poder realizar una comparación visual de los resultados.

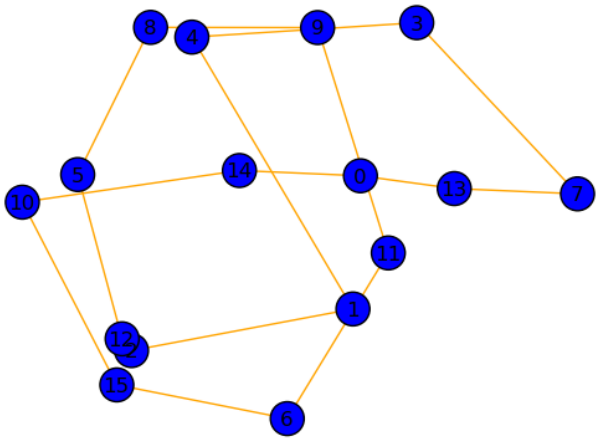

Figura 4.5: 2opt sobre *ánguloGreedy* en instancia angular

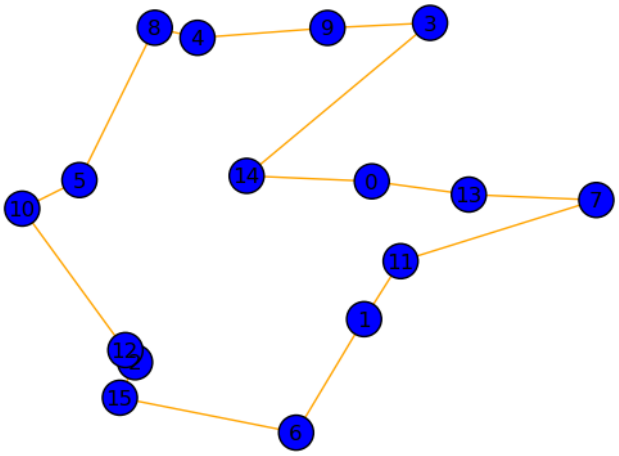

Figura 4.6: 2opt sobre *ánguloGreedy* en instancia TSP

En estas dos imagenes se puede ver como 2-opt se mueve por soluciones muy diferentes al tener una función objetivo que priorice cosas diferentes. En el caso de la instancia angular, 2-opt mantiene una estructura parecida a la generada por *ánguloGreedy* pero suaviza los cambios de dirección bruscos, por ejemplo creando una linea casi sin cambios de dirección entre los nodos 10, 14, 0, 13 y 7.

Por el lado del TSP, se puede ver que la solución que había salido de *ánguloGreedy* no iba por buen camino, ya que ni siquiera era el objetivo de esta heurística, por lo que la solución de 2-opt quedó sustancialmente diferente.

A continuación se presentan los ciclos 2-opt, teniendo como entrada los resultados de *farthestInsertion*.

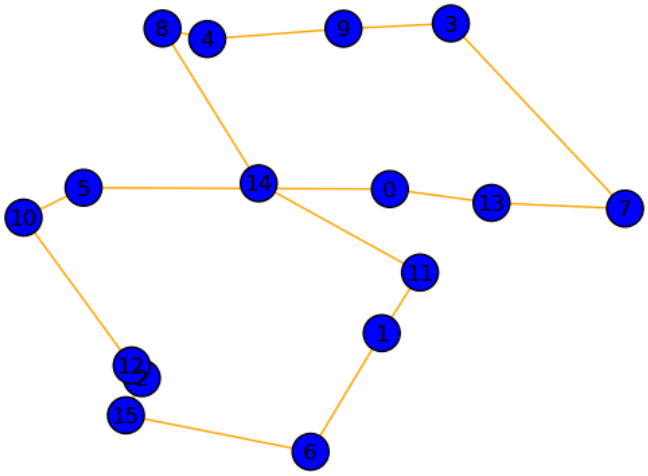

Figura 4.7: 2opt sobre *farthestInsertion* en instancia angular

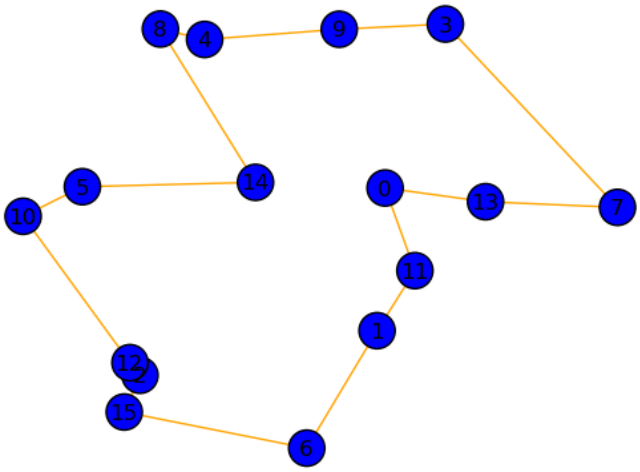

Figura 4.8: 2opt sobre *farthestInsertion* en instancia TSP

De las imagenes anteriores se puede ver que 2-opt intenta nuevamente alinear varios puntos en el caso angular para que los giros sean lo menor posible. Mientras que en el caso del TSP, 2-opt no dió una solución mejor que la que ya se había obtenido por la heurística inicial.

4.4. Búsqueda Local: 3-opt

Al igual que 2-opt, 3-opt es una heurística de búsqueda local que tiene como objetivo mejorar un ciclo actual, también revisando sucesivas vecindades hasta caer en un mínimo local. Como su nombre lo indica, en este caso la vecindad de un ciclo viene dada no por borrar 2 ejes y reconstruir el ciclo, sino 3 ejes. Esto hace que la reconstrucción no sea única, lo que quiere decir que para visitar toda la vecindad no solo hay que elegir todas las ternas de ejes posibles, sino que por cada terna hay que rearmar el ciclo de todas las formas posibles.

Luego, de la misma manera que para 2-opt, la idea de esta búsqueda local será aplicarla sobre la salida de las heurísticas constructivas presentadas anteriormente.

A continuación, solo se presentará el ciclo resultado de 3opt para el caso en donde el problema es angular y el ciclo de entrada es el resultado de *ánguloGreedy*. Solo se presenta este caso debido a que los demás casos que se estaban analizando visualmente dieron el mismo resultado que 2-opt.

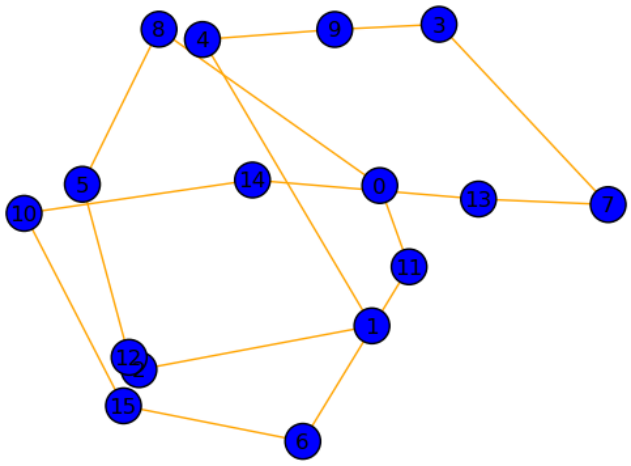

Figura 4.9: 3opt sobre *ánguloGreedy* en instancia angular

Nuevamente se puede ver que lo que hace está búsqueda local es mantener la misma estructura que el ciclo que le fue pasada como parámetro pero tratando de alinear varios de los puntos.

El hecho de utilizar la búsqueda local 2-opt, y luego la búsqueda local 3-opt, induce a pensar lógicamente en esta misma estrategia de búsqueda pero con más opciones de intercambios de ejes. En general, la literatura recomienda utilizar 3-opt como un buen balance entre la calidad de los resultados obtenidos y el esfuerzo computacional utilizado. Utilizar búsquedas k-opt con k más alto, no solo significa una mayor cantidad de grupos de ejes a intercambiar, sino que por cada uno de estos grupos existen más combinaciones posibles para rearmar el ciclo, lo que hace que la complejidad temporal de ejecución se incremente rapidamente. Es por esta razón, que se decidió realizar pruebas solo con 2-opt y 3-opt.

Otro hecho interesante a notar es que, según como esté codificada, 3-opt puede comportarse como 2-opt también, si en el intercambio de los ejes se permite que uno quede en el lugar que estaba originalmente. Esto en la práctica no necesariamente induce una mejor solución, dado que usualmente a estas heurísticas se les suele dar un tiempo máximo de ejecución y 2-opt puede llegar a conseguir mejores soluciones porque es capaz de recorrer más vecindades en el tiempo de ejecución prefijado.

4.5. Comparación de heurísticas iniciales

En las secciones anteriores se presentaron las heurísticas desarroladas para ser utilizadas como iniciales. Para ver cuál de las diferentes opciones es la más conveniente, se realizaron pruebas sobre las instancias random que ya se estuvieron trabajando en etapas anteriores. En este caso solo importó obtener la calidad del ciclo originado por cada una de las opciones. Es cierto que aunque un ciclo sea peor que otro, puede resultar en un mejor desempeño porque puede llegar a quedar un árbol en el que se necesite recorrer menos nodos para resolver el problema hasta la optimalidad. Sin embargo, el objetivo de las heurísticas iniciales es obtener la mejor cota primal inicial posible,

asumiendo que un buen impacto sobre esta primer cota se reflejará en un mejor comportamiento del algoritmo en su totalidad. Es por esto que para comparar las diferentes combinaciones de heurísticas y búsquedas locales se comparó solamente la calidad de la cota primal encontrada.

A continuación se presentan los resultados sobre 6 instancias random y diferentes combinaciones de alfa y beta, siempre sumando 1.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	10982	10992.1	11002.2	11012.3	11022.4
FarthestInsertion	6874	7693.13	8403.27	9113.4	9823.54
2-optÁngulo	6964	7693.13	8080.41	10017.5	10865.2
2-optFarthest	6874	7693.13	8270.5	8657.25	9043.99
3-optÁngulo	6964	7693.13	9768.44	9921.22	10764.4
3-optFarthest	6874	7693.13	8270.5	8657.25	9043.99

Cuadro 4.1: Ciclos iniciales para Instancia Random 1

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	22019	22811.7	23604.5	24397.3	25190
FarthestInsertion	10957	16831.4	19792.4	21458.1	25720.3
2-optÁngulo	12309	16743.9	18262.5	24397.3	24871.3
2-optFarthest	10759	15364.9	16779.6	19388	23549.7
3-optÁngulo	10728	14527.3	17589	19398.7	22501.8
3-optFarthest	10759	14905	16779.6	18711.7	21995.8

Cuadro 4.2: Ciclos iniciales para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	13650	13789	13928.1	14067.1	14206.1
FarthestInsertion	7934	10540.9	12415.8	14290.7	16165.5
2-optÁngulo	8825	11324.5	12118.9	13020.8	14206.1
2-optFarthest	7934	10324.1	11553.4	12385.7	12782.5
3-optÁngulo	7754	11287.8	11096	12092.6	13708.8
3-optFarthest	7934	10183	11096	12128	12782.5

Cuadro 4.3: Ciclos iniciales para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	9137	9115.75	9094.5	9073.25	9052
FarthestInsertion	6435	7570.27	8449.32	9177.98	9906.65
2-optÁngulo	7300	8095.46	8949.89	8502.04	9052
2-optFarthest	6435	7306.8	8449.32	8292.53	9716.17
3-optÁngulo	6473	7306.8	7868.69	8292.53	9052
3-optFarthest	6311	7306.8	7868.69	8292.53	9716.17

Cuadro 4.4: Ciclos iniciales para Instancia Random 4

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	18736	20343.2	21585.5	22713.5	23834.3
FarthestInsertion	9865	14258.5	19867.6	20948.7	22502.7
2-optÁngulo	9951	14594.5	17048.4	22287.4	23786.3
2-optFarthest	9673	12927.2	15646.4	17945.7	20284.6
3-optÁngulo	9462	14582.3	15824	17314.4	20411.5
3-optFarthest	9673	12927.2	15551.9	15779.2	18271.4

Cuadro 4.5: Ciclos iniciales para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
ÁnguloGreedy	20415	21410.5	22245.1	23079.6	23914.2
FarthestInsertion	11666	18579.9	25014.9	25282.3	26759
2-optÁngulo	12054	17838.6	21336.6	23041.3	23854.8
2-optFarthest	11666	18068.7	20397.5	21671.5	23647
3-optÁngulo	11947	17458	21336.6	21054.9	23536.2
3-optFarthest	11666	18068.7	20224.3	21917.2	23049.3

Cuadro 4.6: Ciclos iniciales para Instancia Random 6

De los resultados anteriores se puede ver que, si bien no siempre FarthestInsertion es mejor que ÁnguloGreedy, sí suele haber un dominio cuando se utiliza la búsqueda local 3-opt luego. En 20 de los 30 casos presentados la combinación FarthestInsertion con 3-opt fue la que generó un mejor ciclo, en segundo lugar se encuentra la combinación ÁnguloGreedy con 3-opt con 12 casos, pero con la particularidad de que 5 de ellos son compartidos con FarthestInsertion con 3-opt. Más atrás se encuentran las restantes combinaciones.

Algo también importante a destacar es que en la mayoría de las instancias más angulares, cuando alfa es 0.25 o 0, la mejor combinación es también FarthestInsertion con 3-opt. Se destaca esto ya que las instancias más angulares fueron las que presentaron mayor desafío en los estudios previos.

Dados los resultados presentados, se decidió continuar con el estudio del problema adoptando la combinación FarthestInsertion con 3-opt como heurística inicial para el algoritmo, debido a que encontró la mayoría de las mejores soluciones factibles. Cabe destacar que los tiempos de

ejecución son casi indistinguibles entre las diferentes combinaciones de heurísticas y búsquedas locales cuando estos se analizan porcentualmente respecto del tiempo total tomado por el branch-and-cut, por lo que no resulta un aspecto a considerar en la comparación.

Capítulo 5

Estudio Poliedral

En esta sección el objetivo es poder entender de mejor manera el conjunto de soluciones factibles para poder obtener más información que sea de utilidad al resolver cada instancia particular.

Las heurísticas iniciales utilizadas hasta el momento mostraron un comportamiento razonable, incluso llegando al valor óptimo en algunas instancias con pocos nodos. Sin embargo, aún en estas instancias pequeñas, el algoritmo utilizado hasta el momento no logra resolver muchos de los casos en el tiempo fijado dado que, si bien se tiene como cota primal al óptimo real, las cotas duales son muy débiles y no se logra llegar al certificado de optimalidad.

Es por lo dicho anteriormente que el objetivo de la sección se basa en encontrar más información sobre el poliedro asociado al conjunto de soluciones factibles para poder obtener cotas duales de mayor calidad y así poder resolver las instancias hasta la optimalidad. Para esto, se estudiarán igualdades y desigualdades válidas para el conjunto de soluciones factibles que sirvan para separar puntos no enteros que se encuentren en las relajaciones lineales.

Si bien las igualdades y desigualdades válidas encontradas pueden ser muy útiles a la hora de encontrar mejores relajaciones lineales en cada nodo del árbol del branch-and-cut, se debe considerar la cantidad de restricciones que se están agregando al modelo ya que esto puede tener consecuencias en el tiempo de resolución de la relajación, pudiendolo incrementar considerablemente. Es importante realizar un balance entre los beneficios en las cotas duales que surgen de agregar estas restricciones válidas, y los tiempos que tardan las relajaciones en los modelos con más restricciones.

5.1. Igualdades del sistema minimal

Si bien tener n variables para un modelo genera un espacio de la misma dimensión, el poliedro asociado al conjunto de soluciones factibles puede tener dimensión menor a n. Esto sucede cuando existen igualdades que se cumplen para todos los puntos del poliedro en cuestión.

Una desigualdad en un modelo se traduce en un hiperplano que divide al espacio original en dos nuevos semiespacios. Dependiendo de la orientación de la desigualdad será uno o el otro semiespacio el que se debe considerar.

Sin embargo, al tener una igualdad en un modelo, todos los puntos factibles tienen que cumplirla, lo que se traduce en un hiperplano que, a diferencia de las desigualdades, tiene la particularidad de que todos los puntos factibles se encuentran dentro del mismo. Esto hace que todos los puntos se encuentren en un espacio de dimensión menor a la original. *A priori*, fijando solamente una igualdad, la dimensión a considerar pasa a ser una menos que la dimensión original.

Luego, cada igualdad válida nueva, donde nueva en este caso se entiende como que no puede ser deducida de las anteriores presentes, genera un nuevo hiperplano donde se encuentran todos los puntos factibles, reduciendo en uno la dimensión.

Es interesante notar que todo poliedro se puede escribir mediante un *sistema minimal* de igualdades y las desigualdades válidas que generan la cápsula convexa del modelo. Las igualdades

son un conjunto linealmente independiente que reducen una a una la dimensión del poliedro en cuestión, respecto de la dimensión original del espacio a donde pertenecen las variables. Cada igualdad del sistema minimal es un hiperplano donde se encuentran todos los puntos factibles, y finalmente el poliedro pertenece al espacio generado por la intersección de todos los hiperplanos generados por las igualdades. Una vez que se tiene este espacio de menor dimensión, se tienen todas las desigualdades necesarias para describir la cápsula convexa del conjunto de puntos factibles.

Teóricamente, poder conocer el conjunto de igualdades válidas del sistema minimal es interesante porque reduce al mínimo posible la dimensión del espacio de búsqueda de soluciones. Computacionalmente, tener este conjunto de igualdades puede devenir en relajaciones notoriamente más fuertes, ya que se estarían descartando grandes conjuntos de soluciones que no vale la pena tener en cuenta. Sin embargo, no se debe dejar de lado el hecho de que agregar todas las igualdades del sistema minimal al modelo puede generar relajaciones lineales más pesadas para cada uno de los nodos del branch-and-cut.

A continuación se presentan las diferentes familias de igualdades halladas para la formulación estudiada, así como también se conjentura un resultado sobre la dimensión de la cápsula convexa del problema.

Para la búsqueda de estas igualdades válidas se utilizaron diferentes caminos. En primer lugar, hay igualdades válidas que ya se derivan del TSP y ya se encuentran plasmadas en el modelo original. Por otro lado, algunas de las igualdades se generaron a partir de la idea intuitiva de lo que representa cada variable en el problema real, para solucionar ciertos problemas particulares que se presentaban en el modelo, no porque este no fuese correcto sino porque la relajación admite ciertas soluciones fraccionarias que se pueden evitar. Por último, se utilizó un software llamado PORTA que sirve para poder enumerar las igualdades del sistema minimal y las desigualdades de la cápsula convexa de la formulación para instancias pequeñas, ya que el algoritmo utilizado no es polinomial. La idea de este camino es poder identificar relaciones entre las variables en instancias de pocos nodos, y poder extrapolar estas relaciones a cualquier instancia en general.

Asignación TSP Los modelos de TSP presentados, y luego adaptados para AngTSP, ya contienen dos familias de igualdades presentes que se denominan igualdades de asignación. Las mismas son

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n \tag{5.1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n \tag{5.2}$$

Como ya fue mencionado anteriormente, estas igualdades indican que para cada vértice tiene que haber exactamente un eje de salida y exactamente un eje de entrada.

Estas igualdades ya se encuentran en los modelos para que los mismos sean correctos, por lo que aquí solamente se enuncian como parte del sistema minimal pero no para lograr una mejora en las cotas duales.

Asignación AngTSP Esta familia de igualdades fue motivada por una debilidad de los modelos que se visualizó al realizar los estudios preliminares presentandos anteriormente.

La adaptación original del TSP al AngTSP se basó en agregar las variables binarias de utilización de los ángulos, necesarias para la función objetivo, y luego se agregó la restricción 3.36 para que los modelos fuesen correctos. Esta restricción indica que una variable de ángulo debe ser necesariamente 1 si se elegieron los dos ejes que lo conforman como parte del circuito hamiltoniano resultado. Esta restricción no indica que un ángulo no deba ser utilizado cuando no se utilicen sus dos ejes, ya que la función objetivo tratará siempre de tener las variables de ángulo en 0(ya que

los pesos en la función objetivo son siempre positivos), a menos que exista una restricción que no permita esto.

Esta libertad en el valor de las variables de ángulo generó un notorio problema en la relajación lineal del mismo cuando el problema es puramente angular, es decir cuando todo el peso de la función objetivo se encuentra sobre los ángulos y nada sobre las distancias. Este problema ya fue explicado cuando se presentaron las cotas de las formulaciones estudiadas, y se resume en que, como están hasta ahora las restricciones, la relajación lineal del nodo raíz en el problema angular da 0.

Es por este problema que se pensó esta familia de igualdades que tiene como objetivo no permitir la fijación de todas las variables de ángulo en 0. La familia diseñada se llamó de asignación para el AngTSP ya que tiene una razón de ser análoga a las igualdades de asignación para TSP. Esta familia está dividida en tres tipos de igualdades.

LeftVertex

$$\sum_{j,k=1}^{n} y_{ijk} = 1, \quad i = 1, \ldots, n \tag{5.3}$$

Las igualdades LeftVertex indican que cada vértice debe ser punto izquierdo de exactamente un ángulo. Esta subfamilia ya por sí sola logra descartar la relajación lineal con valor 0.

CenterVertex

$$\sum_{i,k=1}^{n} y_{ijk} = 1, \quad j = 1, \ldots, n \tag{5.4}$$

Análogamente, las igualdades CenterVertex indican que cada vértice debe ser punto medio de exactamente un ángulo.

RightVertex

$$\sum_{i,j=1}^{n} y_{ijk} = 1, \quad k = 1, \ldots, n \tag{5.5}$$

Por último, la familia RightVertex indica que cada vértice tiene que ser punto derecho de exactamente un ángulo.

Si bien las igualdades son bastante simples e intuitivas, sirven para descartar varios puntos fraccionarios que hasta el momento eran factibles en la relajación lineal, como por ejemplo la relajación de valor 0 que motivó la búsqueda de nuevas igualdades.

Existen algunos puntos interesantes a destacar sobre estas familias de igualdades. En primer lugar, cabe notar que la cantidad de restricciones generadas para esta familia pertenecen a $\mathcal{O}(n)$, por lo que no debería ralentizar demasiado la relajación lineal.

Por otro lado, se destaca el hecho de que las subfamilias presentadas no son redundantes entre sí. Si bien es cierto que al considerar todas las igualdades juntan existen algunas que son linealmente dependientes con el resto, no se puede derivar una subfamilia entera a partir de las otras igualdades ya presentes.

Por último, se debe mencionar el hecho de que estas igualdades no solamente están cortando puntos fraccionarios factibles de la relajación lineal, sino que también estan cortando puntos enteros que hasta el momento eran factibles para el modelo. Esto no significa una pérdida de la correción del modelo por la inclusión de estas igualdades, ya que los puntos enteros que se descartan no son óptimos o existen óptimos alternativos, por lo que el modelo sigue siendo correcto. Además de lo explicado anteriormente, se menciona el hecho de que los puntos descartados son puntos que no se corresponden con ideas intuitivas que se tiene sobre el modelo. Dado el significado de las variables, no tiene sentido tener circuitos hamiltonianos en los que haya más ángulos utilizados que los definidos por las variables de eje.

Relación fuerte entre ángulos y ejes Esta familia de igualdades fue motivada por el hecho de tener una relación fuerte entre las variables de ángulos y las variables de ejes. Hasta el momento, la relación que existe en el modelo original es débil en el sentido que ya fue mencionado, que es que solamente se obliga a utilizar la variable de ángulo cuando se utilizan los dos ejes que lo conforman. Es por esto que se presentan las siguientes igualdades que capturan una idea mucho más fuerte de como se relacionan los ejes y los ángulos.

$$\sum_{k=1}^{n} y_{ijk} = x_{ij}, \quad i, j = 1, \ldots, n \tag{5.6}$$

Esta igualdad indica que si se usa el eje ij, entonces tiene que existir exactamente un ángulo que pasa por los puntos i y j, y luego vaya para cualquier otro punto k.

Mediante un razonamiento análogo, se presenta la siguiente igualdad.

$$\sum_{k=1}^{n} y_{kij} = x_{ij}, \quad i, j = 1, \ldots, n \tag{5.7}$$

Con estas igualdades ahora sí se cumple la propiedad recíproca de la que ya fue mencionada en varias oportunidades. Cuando una variable de eje x_{ij} se encuentra en 0, no puede suceder que se esté utilizando una variable de ángulo que incluya a los puntos i y j. De esta manera se logra fortalecer la idea intuitiva sobre lo que significan las variables en un circuito hamiltoniano posible.

Con esta familia de igualdades ya no se tiene una cantidad lineal de restricciones, sino que se tiene una cantidad perteneciente a $\mathcal{O}(n^2)$. La aparición de esta cantidad de restricciones fue un resultado esperado por lo que se había obtenido de manera experimental al buscar el sistema minimal para poliedros de instancias con pocos puntos.

Algo importante a notar es que estas nuevas igualdades sí son linealmente dependientes con las presentadas anteriormente. Se puede demostrar que a partir de las igualdades de asignación de TSP y de las igualdades de relación fuerte, se pueden derivar todas las de asignación de AngTSP presentadas. Sin embargo, no por este motivo se descartan las igualdades de asignación de AngTSP antes de evaluarlas computacionalmente, ya que si bien son más débiles que las de relación fuerte, son una cantidad lineal y eso puede ayudar a los tiempos de ejecución del algoritmo.

Dimensión de la cápsula convexa En esta sección se presenta una conjetura sobre la dimensión del espacio donde vive la cápsula convexa del modelo estudiado. La misma es producto del estudio teórico presentado anteriormente, sumado a la experimentación realizada con el software PORTA para poliedros de instancias de pocos puntos.

Conocer todas las igualdades del sistema minimal no sólo conforma un resultado teórico, sino que puede resultar en mejoras computacionales al fortalecer la relajación lineal del modelo.

Hasta el momento se cuenta con 5 familias de igualdades que se cumplen para cada punto, las de asignación de TSP y las de asignación de AngTSP, sumadas a 2 familias de igualdades que se cumplen para cada par de puntos distintos, ya que las variables de eje o ángulo que consideran a un mismo punto no son relevantes. Esto da un total de $5n + 2n(n-1)$ igualdades, de las cuales ya se sabe que no son todas linealmente independientes.

Luego de obtener dicha cantidad de igualdades válidas se procedió a experimentar computacionalmente, mediante PORTA, para contrastar la cantidad de igualdades obtenidas, la cantidad de igualdades del sistema minimal, y la cantidad de igualdades linealmente independientes de las obtenidas.

A continuación se presentan los resultados obtenidos para poliedros de instancias de entre 4 y 8 puntos.

Puntos \ Conjunto	Obtenidas	Independientes	Minimal
4	44	27	31
5	65	44	58
6	90	65	70
7	119	90	90
8	152	119	119

Cuadro 5.1: Comparación de cantidad de igualdades

La tabla anterior muestra varios puntos a destacar que conforman la base para la conjetura a presentar.

Por un lado se puede ver la relación que hay entre la primera y la segunda columna. La primera columna muestra la cantidad total de igualdades mencionadas anteriormente, las cuales siguen la sucesión $5n+2n(n-1)$. La segunda columna sigue exactamente la misma sucesión pero desplazada en un término, es decir, cuando la fórmula dada se evalúa en n para la primera columna, se evalúa en $n-1$ para obtener los valores de la segunda columna.

Por otro lado, la segunda y la tercera columna coinciden para 7 y 8 puntos, esto da la pauta de que es posible que entre todas las igualdades obtenidas se encuentren todas las del sistema minimal, a partir de 7 puntos. La disidencia que hay con los poliedros de instancias de menos puntos puede ser explicada por el hecho de que al haber pocos puntos hay pocas combinaciones posibles para un tour, y todas estas combinaciones cumplen igualdades que no se cumplen cuando hay más puntos en el problema.

Dadas las evidencias observadas, se conjetura que entre las igualdades explicitadas se incluyen a todas las presentes en el sistema minimal para cuando la cantidad de puntos es mayor o igual a 7 (la demostración o refutación de la misma excede el alcance de este trabajo). De hecho, se puede probar que las igualdades de asignación de AngTSP se consiguen operando entre las igualdades de relación fuerte y las igualdades de asignación de TSP, por lo que se conjentura que las igualdades del sistema minimal se encuentran incluidas entre las de asignación de TSP y las igualdades de relación fuerte.

En los casos de pocos puntos, es decir 6 puntos o menos, hay más igualdades que las obtenidas. Encontrar estas igualdades faltantes no solo conforman un resultado teórico para el sistema minimal de pocos puntos, sino que pueden devenir en resultados prácticos para la resolución algorítmica del problema, ya que en varios casos las igualdades que se cumplen solo en poliedros de instancias de pocos puntos, pasan a ser desigualdades fuertes de los poliedros del mismo problema pero con más puntos.

Es por esto que se buscó caracterizar todas las igualdades para cualquier cantidad de puntos, asumiendo la conjetura sobre las familias de igualdades ya mencionadas anteriormente.

A continuación, se presentan las igualdades faltantes para los poliedros del problema con 4, 5 y 6 puntos. Casi todas las igualdades presentadas luego se explayarán con mayor detalle en la sección de desigualdades válidas.

Igualdades faltantes 4 puntos En el caso de 4 puntos, las igualdades faltantes provienen del simple hecho de que si se usan 3 puntos seguidos en una solución entonces en la solución seguro que antes está el punto restante, y después también. Esto se traduce en las siguientes igualdades

$$y_{ijk} = y_{rij} \tag{5.8}$$

$$y_{ijk} = y_{jkr} \tag{5.9}$$

Donde i, j, k y r son todos índices diferentes.

Estas nuevas relaciones generan, para el caso de 4 puntos, 24 nuevas igualdades que en conjunto con todas las igualdades mencionadas anteriormente conforman las 31 igualdades lineamente independientes presentes en el sistema minimal del poliedro proveniente del problema con 4 puntos.

Si bien estas igualdades son válidas solamente en 4 puntos, hay formas de transformarlas en desigualdades válidas para cualquier instancia del problema de mayor cantidad de puntos.

Igualdades faltantes 5 puntos El caso de 5 puntos es en el que más igualdades faltan, hasta el momento se obtuvieron 44 igualdades linealmente independientes, y el sistema minimal obtenido computacionalmente indica un valor de 58 igualdades.

Una primera familia válida de igualdades para 5 puntos, proviene del hecho de que dados dos puntos i y j, siempre tiene que existir exactamente una variable, ya sea x o y, que las relacione en la solución. Siendo sólo 5 puntos, siempre sucede que i está 1 o 2 posiciones antes que j, o 1 o 2 posiciones despues j. Esto da lugar a la siguiente familia de igualdades

$$x_{ij} + \sum_{k=1,k\neq i,k\neq j}^{n} y_{ikj} + x_{ji} + \sum_{k=1,k\neq i,k\neq j}^{n} y_{jki} = 1 \tag{5.10}$$

Para todo par de i y j diferentes.

Estas igualdades pueden ser fácilmente transformadas en desigualdades para instancias del problema con más puntos, y serán explayadas cuando se traten las desigualdades válidas.

Esta nueva familia de igualdades para 5 puntos, sumadas a las obtenidas anteriormente para cualquier cantidad de puntos, conforman un total de 49 igualdades linealmente independiente sobre las 58 que tiene el sistema minimal.

La siguiente familia de igualdades válidas para 5 puntos provienen del hecho de que si se toma un punto cualquiera como el punto medio del ciclo resultado, entonces los otros cuatro puntos restantes se dividen en dos grupos de dos nodos, un grupo son los que están antes del punto medio del ciclo, y se encuentran antes en algún orden posible; mientras que el otro grupo de dos nodos son los que se encuentran detrás del punto medio del ciclo, también en algún orden posible.

$$y_{abc} + y_{bac} = y_{cde} + y_{ced} \tag{5.11}$$

Donde a, b, c, d y e son 5 índices diferentes.

De esta manera, se obtiene un total de 54 igualdades lineamente independientes sobre las 58 buscadas.

Por último, las 4 igualdades linealmente independientes restantes provienen de una familia *ad hoc* de igualdades para 5 puntos, que es difícil de caracterizar intuitivamente y que no resultaron de utilidad para el modelo ya que no se extendieron a una mayor cantidad de puntos. Es por esto que no se expresa la caracterización de esta familia pero sí se nota que con estas nuevas igualdades se obtiene un total de 58 igualdades lineamente independientes que conformarían la descripción completa del sistema minimal para 5 puntos.

Igualdades faltantes 6 puntos Para el caso de 6 puntos, son solo 5 igualdades las que faltan para obtener la misma cantidad que las que se obtuvieron mediante PORTA. Estas igualdades provienen del hecho de que las variables y predican sobre el ángulo que se forma con 3 puntos, y como se está en el caso de 6 puntos, si hay una variable y que se esté utilizando para relacionar 3 puntos en algún orden dado, entonces tiene que existir una variable y que relacione en algún orden a los otros 3 puntos restantes. Este hecho se traduce en la siguiente familia de igualdades

$$y_{abc} + y_{acb} + y_{bac} + y_{bca} + y_{cab} + y_{cba} = y_{ijk} + y_{ikj} + y_{jik} + y_{jki} + y_{kij} + y_{kji} \tag{5.12}$$

Donde todos los índices corresponden a puntos diferentes.

Sumando estas igualdades a las obtenidas anteriormente, se consiguen las 70 igualdades linealmente independientes para el caso de 6 puntos. Estas igualdades pueden ser convertidas en desigualdades para casos de más puntos, lo cual será explayado en la sección de desigualdades válidas.

De esta manera, parece tenerse caracterizado el conjunto de igualdades del sistema minimal para instancias de cualquier cantidad de puntos mayor o igual a 4.

Experimentación En esta sección se muestran los resultados obtenidos en la experimentación de la inclusión de las igualdades válidas presentadas en el modelo.

Dado que todas las familias generales presentadas conforman una cantidad de igualdades pertenencientes a $\mathcal{O}(n)$ o $\mathcal{O}(n^2)$, en cada combinación de igualdades estudiada se opta por incluir en el modelo desde un comienzo las igualdades pertinentes a esa combinación. Como las igualdades de orden cuadrático pueden resultar muchas, es necesario realizar un estudio por separado de las diferentes familias para poder hacer un balance entre los beneficios que se obtienen al incluir las igualdades, contra las desventajas que se pueden derivar, por ejemplo por convertir la relajación lineal del problema en un problema muy pesado.

A continuación se muestran los resultados obtenidos de la experimentación sobre las 6 instancias random con 5 combinaciones diferentes de parámetros *alfa* y *beta*, a cada combinación se le otorgó un tiempo máximo de ejecución de 1800 segundos. En los casos donde se consume el tiempo máximo de corrida, se reporta entre parentesis el gap porcentual obtenido.

Todas estas pruebas se realizaron sobre las siguientes tres combinaciones

- El modelo denominado *Sin restricciones* es el modelo original antes de realizar el estudio poliedral, por lo que en lo que respecta a igualdades solamente tiene a las de asignación de TSP.
- El modelo denominado *Con restricciones lineales* es el modelo original sumado a todas las igualdades que pertenecen a $\mathcal{O}(n)$. Es decir que se tiene sumadas las igualdades de asignación de AngTSP.
- El modelo denominado *Con restricciones cuadráticas* es el modelo original sumado a todas las igualdades que pertenecen a $\mathcal{O}(n^2)$. Es decir que se tiene sumadas las igualdades de relación fuerte entre las x y las y.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	0.07	8.49	482.73	(33.71 %)	(53.90 %)
DFJPRLinRes	17.8	4.96	11.32	80.91	(4.81 %)
DFJPRCuadRes	0.06	0.08	0.18	0.14	1.9

Cuadro 5.2: Tiempos para Instancia Random 1

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	3.94	(29.47 %)	(50.34 %)	(78.24 %)	(100 %)
DFJPRLinRes	(9.70 %)	(20.53 %)	(26.81 %)	(45.87 %)	(52.71 %)
DFJPRCuadRes	1.68	404.33	(1.98 %)	826.78	893.79

Cuadro 5.3: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	0.33	1645.53	(31.68 %)	(64.91 %)	(95.79 %)
DFJPRLinRes	37.21	367.09	(13.36 %)	(27.69 %)	(42.56 %)
DFJPRCuadRes	0.29	6.36	45.12	365.96	251.19

Cuadro 5.4: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	0.02	2.86	46.24	378.9	416.83
DFJPRLinRes	1.87	1.81	3.07	11.36	183.35
DFJPRCuadRes	0.03	0.04	0.04	0.05	0.23

Cuadro 5.5: Tiempos para Instancia Random 4

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	2.09	(14.96 %)	(46.61 %)	(71.64 %)	(100.00 %)
DFJPRLinRes	417.4	(7.53 %)	(27.46 %)	(37.57 %)	(62.28 %)
DFJPRCuadRes	2.4	20.92	1440.5	570.07	1497.13

Cuadro 5.6: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
DFJPRSinRes	17.14	(29.76 %)	(56.44 %)	(78.42 %)	(100.00 %)
DFJPRLinRes	(16.67 %)	(29.11 %)	(41.87 %)	(52.42 %)	(64.59 %)
DFJPRCuadRes	5.2	(3.27 %)	(12.37 %)	(14.65 %)	(19.91 %)

Cuadro 5.7: Tiempos para Instancia Random 6

De los resultados anteriores se desprenden varios puntos de interés a analizar.

En primer lugar, es notoria la dismunición en los tiempos de ejecución en la mayoría de las instancias al incluir las igualdades cuadráticas encontradas. Aún las instancias que siguen siendo difíciles para el nuevo modelo, las que no terminan dentro de la media hora fijada, finalizan su ejecución con un gap considerablemente mejor que el obtenido por el modelo original.

Se puede observar que hay varias combinaciones de instancias y parámetros que antes no eran resolubles por el modelo y ahora sí lo son, incluso en algunos casos las instancias se resuelven en tiempos muy bajos. Un punto que se nota de todas las experimentaciones, es que el problema se torna bastante más difícil conforme se le da más importancia a los ángulos por sobre las distancias. De hecho, antes de incluir las igualdades, el problema puramente angular solo se podía resolver para la instancia más pequeña, la instancia número 4 que consta de 12 nodos; ahora en casi todas

las instancias se resuelve el problema puramente angular, excepto la instancia con mayor cantidad de nodos entre las utilizadas.

Otro punto interesante para notar, es el hecho de poder ver para cuáles igualdades sí fue beneficiosa su inclusión. De los resultados parecería desprenderse que la sola inclusión de las igualdades lineales conforman un beneficio para la resolución del problema, pero en algunas instancias agregan un *overhead* en la resolución de las relajaciones lineales que deriva en un peor desempeño respecto del modelo original.

Por otro lado, la inclusión de las igualdades cuadráticas parece ser una inclusión más que razonable, resultando en mejoras considerables para los tiempos de ejecución de cada instancia, y cuando se consume el tiempo total de corrida, se puede observar que los gaps obtenidos son mucho más chicos que los originales.

Dados los resultados obtenidos, se optó por continuar el estudio del problema con el modelo original aumentado por las igualdades cuadráticas de relación fuerte entre las variables x e y.

5.2. Desigualdades Válidas

En esta sección el objetivo es presentar las desigualdades válidas estudiadas para fortalecer la formulación del problema mediante planos de corte. Se mostrarán las desigualdades válidas halladas, justificando la inclusión de las mismas como tales, discutiendo la fortaleza de cada una de ellas. Para esto, se utilizó nuevamente PORTA, ya que permite conocer la dimensión de una desigualdad para poliedros de instancias de pocos puntos; y luego se conjeturan extrapolaciones de los resultados provistos a poliedros de instancias de mayor cantidad de puntos.

Luego se presentará la experimentación realizada con las mismas, para poder analizar como se comporta cada una de las familias halladas en el algoritmo branch-and-cut.

Para todas las desigualdades presentadas se asume que los índices utilizados en las variables son diferentes ya que las variables que tengan como índice dos o más veces el mismo punto no están definidas.

Desigualdades *Todo orden 3 puntos*

Formulación Esta familia de desigualdades se enuncia de la siguiente forma

$$y_{ijk} + y_{ikj} + y_{jik} + y_{jki} + y_{kij} + y_{kji} \leq 1 \qquad \forall i, j, k = 1 \dots n \tag{5.13}$$

Interpretación Esta desigualdad surge a partir de la igualdad que fue utilizada para completar el sistema minimal para el problema con 6 puntos. Basicamente lo que significa esta desigualdad es que dados 3 puntos existe una sola variable de ángulo que los relacione a los 3, y en muchos casos no será ninguna. Es por esto que la desigualdad indica que la suma de todos los ángulos posibles de los 3 puntos no puede superar 1.

Dimensión Si bien la desigualdad está intimamente relacionada con una igualdad para el poliedro de 6 puntos, al hacer crecer la cantidad de puntos la dimensión de la cara definida por la desigualdad comenzo a alejarse de la dimensión del poliedro probando que no es faceta. Esto, en general, suele concluir en el hecho de que la desigualdad no será lo suficientemente fuerte como para obtener buenos planos de corte.

Separación El problema de separación para esta desigualdad puede ser resuelto de forma exacta ya que un algoritmo de separación por fuerza bruta tiene una complejidad perteneciente a $\mathcal{O}(n^3)$, lo cual no resulta prohibitivo para la cantidad de puntos que se están utilizando en las instancias de prueba. En el caso de que el objetivo sea resolver instancias con cantidad de puntos de mayor orden, es problable que se tenga que modificar la separación de esta familia para no ralentizar el proceso de la búsqueda de planos de corte.

Desigualdades *Todo orden IJ*

Formulación La familia *Todo orden IJ* está dada por la siguiente desigualdad

$$x_{ij} + \sum_{k=1}^{n} y_{ikj} + x_{ji} + \sum_{k=1}^{n} y_{jki} \leq 1 \qquad \forall i, j = 1 \dots n \tag{5.14}$$

Interpretación Esta familia de desigualdades surge de pensar todos los órdenes posibles para dos puntos dados, acotados a las variables que se tienen en el modelo que solo dejan ver una vecindad de distancia 1 o 2 en el tour resultado. Dados dos puntos i y j, a lo sumo puede pasar una de las siguientes cosas

- i está inmediatamente antes de j, lo cual se ve reflejado en la variable x_{ij}.
- j está inmediatamente antes de i, lo cual se ve reflejado en la variable x_{ji}.
- i está dos lugares antes que j, lo cual se ve reflejado en alguna variable y_{ikj}.
- j está dos lugares antes que i, lo cual se ve reflejado en alguna variable y_{jki}.

Esta situación tiene la salvedad de que no sucede para instancias con menos de 5 puntos, ya que con, por ejemplo 4 puntos, puede suceder que i este dos posiciones antes j y a la vez dos posiciones despues, dado que es un ciclo.

Dimensión En esta familia de desigualdades, lo primero a destacar sobre la dimensión, que ya fue mencionado anteriormente, es que para las instancias con 5 puntos, la cara definida por la desigualdad tiene la misma dimensión que el poliedro en sí mismo porque para 5 puntos siempre se cumple exactamente una de las 4 situaciones descriptas anteriormente. Esta situación, además de poder ser interpretado intuitivamente, es fácil de demostrar mediante una simple enumeración de los circuitos hamiltonianos posibles para 5 puntos. Es por esto que esta familia pasa a ser una familia de igualdades para 5 puntos y forma parte del sistema minimal de la cápsula convexa de dicho problema.

Por otro lado, pasando a poliedros formados por una mayor cantidad de puntos, se pudo ver que para 6, 7 y 8 puntos la dimensión de la cara definida por la desigualdad es exactamente una menos que la dimensión de la cápsula convexa de dichos casos. Este hecho muestra que esta familia de desigualdades es una familia generadora de facetas para los casos mencionados y, dada la estabilidad conjeturada para la dimensión de la capsula convexa a partir de 7 puntos, sería consecuente pensar que esta familia de desigualdades sea una familia generadora de facetas para el problema con cualquier cantidad de puntos mayor o igual a 6.

Separación Al igual que la familia presentada anteriormente, un algoritmo de separación por fuerza bruta para esta familia tiene una complejidad temporal perteneciente a $\mathcal{O}(n^3)$. Por lo que esta opción será la elegida para llevar a la práctica dado que el tamaño de las instancias que se están intentado resolver lo permite.

Desigualdades *En algún lado va l*

Formulación Esta familia de desigualdades está dada por la siguiente relación

$$y_{ijk} \leq y_{lij} + y_{jkl} + \sum_{r,s=1}^{n} y_{rls} \quad \forall i, j, k, l = 1 \dots n \tag{5.15}$$

Interpretación Esta familia de desigualdes trata de capturar la noción de donde se encuentra un cuarto punto, dado que ya se sabe que se utilizaron otros 3. Es decir, si i, j y k no se utilizan en ese orden, o sea la variable y_{ijk} es 0, entonces la desigualdad vale trivialmente y no hay otra cosa para decir. Si, por el contrario, la variable y_{ijk} tiene valor 1, es porque está significando que i, j y k son consecutivos en el circuito representado. Luego, tomando un cuarto punto l siempre distinto a los anteriores, tiene que suceder alguno de los siguientes hechos

- l está justo antes de estos 3 puntos, lo que se ve reflejado en la variable y_{lij}.
- l está justo después de estos 3 puntos, lo que se ve reflejado en la variable y_{jkl}.
- l está en cualquier otro lado del tour por lo que seguro existen dos puntos cualesquiera que *encierran* a l, lo que se ve reflejado en alguna de todas las variables y_{rls}.

Estas situaciones posibles tienen el detalle de que para 5 puntos la desigualdad vale pero la misma no presenta una sumatoria ya que no hay una cantidad de puntos suficientes. Esto no representa ningún problema ya que es lógico ver que para 5 puntos la desigualdad se cumple extrayendo la sumatoria, dado que si ya se sabe que se utilizaron 3 puntos en orden, entonces cualquier cuarto punto que se tome estará o bien justo antes, o bien justo despues de los 3 puntos mencionados.

Por otro lado, para 4 puntos la desigualdad se cumple nuevamente sin la sumatoria ya que el lado derecho debería dar exactamente dos.

Dimensión Para esta familia de desigualdades se verificó la dimensión de la cara pertinente para los poliedros de 4, 5, 6, 7 y 8 puntos. En todos los casos la dimensión buscada fue una menos que la dimensión del poliedro en cuestión. Por lo que, al igual que sucedió con la familia anterior, se cree que esta familia de desigualdades es una familia que induce facetas para el poliedro de cualquier cantidad de puntos, aunque esto sólo haya sido probado por extensión hasta 8 puntos.

Separación El problema de separación para esta familia de desigualdades es más complicado que los analizados anteriormente. En este caso, las desigualdades tienen 6 variables por lo que se debe tener más cuidado a la hora de pensar en resolver este problema por un algoritmo de fuerza bruta. Para la cantidad de nodos que se está intentado resolver, correr un algoritmo con complejidad temporal perteneciente a $\mathcal{O}(n^6)$ en cada nodo del branch-and-cut puede ser completamente prohibitivo, y también atenta contra el objetivo natural de tratar de resolver instancias mayores.

Sin embargo, al algoritmo obvio de fuerza bruta se le puede hacer una pequeña modificación que disminuye considerablemente los tiempos de corrida, logrando una complejidad temporal más cercana a la que ya se había discutido en los anteriores problemas de separación. La modificación se basa en explotar el hecho de que para violar la desigualdad la parte izquierda debería ser mayor que la parte derecha, pero la parte izquierda está conformada por una sola variable de ángulo. Luego, la modificación que se utiliza sobre el algoritmo de fuerza bruta es preguntar si la desigualdad está violada solamente para los casos en donde y_{ijk} es mayor a cierta tolerancia. Si la tolerancia utilizada es mayor a 0 entonces es posible que no se revise una desigualdad que si está violada, pero es un riesgo que se corre en pos de un mejor tiempo de ejecución, sumado

al hecho de que muchas veces se quieren agregar desigualdades que están más violadas que cierta tolerancia porque se cree que corta más potenciales soluciones fraccionarias.

Lo importante para destacar sobre esta pequeña modificación al algoritmo de fuerza bruta es que, dado que las variables x son siempre menores o iguales a 1 en la relajación, y dado que se tienen la relación fuerte entre las variables x y las y, no puede haber demasiadas variables y con valor mayor a la tolerancia en la relajación lineal, lo que hace que se descarte una cantidad considerable de posibilidades. Esta vaguedad en cuanto a la cantidad de variables y que son mayor a la tolerancia puede ser calculada en función de la cantidad de puntos de la instancia y de la tolerancia utilizada. Pero a los efectos prácticos la decisión es utilizar una toleracia que arroje tiempos razonables de ejecución.

Desigualdades *Algún origen*

Formulación Esta familia de desigualdades está dada por la siguiente restricción

$$y_{ijk} \leq \sum_{r=1, r\neq k}^{n} y_{rij} \quad \forall i, j, k = 1 \dots n \tag{5.16}$$

Interpretación Lo que indica esta desigualdad es que si los puntos i, j y k son consecutivos en la solución, entonces tiene que existir algún r de donde se provenga antes de pasar por ij. En el caso de que no se use y_{ijk} la desigualdad se cumple trivialmente.

Dimensión Estas desigualdades fueron testeadas mediante PORTA para conocer la dimensión de la cara que definen, en el caso de 4 puntos esta restricción se cumple por igualdad por la poca combinación de los órdenes entre los puntos posibles. Para 5, 6, 7 y 8 puntos, la dimensión de la cara definida siempre fue una menos que la dimensión del poliedro original, por lo que se cree que las caras definidas por estas desigualdades son facetas para todos los poliedros de más de 4 puntos.

Separación El problema de separación para esta familia puede nuevamente ser atacado desde un enfoque de fuerza bruta. El algoritmo puro de fuerza bruta tiene una complejidad temporal perteneciente a $\mathcal{O}(n^4)$, ya que por cada trío de variables, se realiza la cuenta con la sumatoria del lado derecho de la desigualdad. Sin embargo, al igual que en la familia *en algún lado va l*, del lado izquierdo de la desigualdad sólo se tiene una variable, así que el lado izquierdo solamente se evaluará cuando la variable y_{ijk} tenga un valor mayor a una cierta tolerancia, haciendo que sean muchos menos los casos en los que se tenga que evaluar la sumatoria.

Desigualdades *Algún destino*

Formulación La formulación de esta familia es la siguiente

$$y_{ijk} \leq \sum_{r=1, r\neq i}^{n} y_{jkr} \quad \forall i, j, k = 1 \dots n \tag{5.17}$$

Interpretación En este caso, la interpretación es parecida a la familia anterior pero ahora la desigualdad dice que si se usan los puntos i, j y k consecutivamente, entonces tiene que existir algún r a donde el ciclo se dirige luego.

Dimensión y Separación Tanto el análisis de la dimensión de la cara definida por cada desiguldad, como el algoritmo para resolver el problema de separación, son análogos a los de la familia *Algún origen*

Algo importante para notar de todas las familias explicadas anteriormente, es que cada una de ellas conforma desigualdades que dejan puntos fraccionarios, que antes eran factibles, fuera del poliedro, por lo que su inclusión a priori no es en vano.

Para poder ver que esto es cierto se lo puede demostrar computacionalmente, si se encuentra un caso en el que la desigualdad a analizar deje un punto afuera, ya alcanza para demostrar que la desigualdad no es superflua.

Luego, lo que se hizo para ver esto fue tomar una desigualdad cualquiera de cada familia, lo cual no pierde generalidad ya que en todas las familias explicadas no existen nodos distinguidos. Una vez tomada esta desigualdad, se la introduce al modelo obtenido hasta el momento como función objetivo, y se busca maximizarla. Si la maximización de la desigualdad como función objetivo da un valor mayor estricto al lado derecho de la desigualdad, entonces quiere decir que existe un punto que hasta el momento es factible y que viola dicha desigualdad. Cabe notar que cuando se habla de superar al lado derecho de la desigualdad se está pensando en una versión modificada de la misma, en donde del lado izquierdo se encuentran todas las variables y del lado derecho se encuentran las constantes, o un 0 en el caso que no hubiese.

Otro punto a destacar es que cada desigualdad analizada fue violada en la primera prueba realizada, con lo cual induce a pensar que los puntos que violan estas desigualdades no son puntos muy particulares, sino que están presentes en muchas instancias del problema.

Experimentación En esta sección se presentará la experimentación realizada con las diferentes familias de desigualdades válidas para poder analizar los beneficios de sus inclusiones como planos de corte en el modelo.

En primer lugar se realizó un estudio cuantitativo sobre las mismas 6 instancias random que ya se utilizaron en experimentaciones anteriores. Se hizo este estudio preliminar para poder contar la cantidad de cortes que cada familia introduce al modelo, para poder ver si existen desigualdades que teóricamente sirven pero en la práctica no demasiado.

El estudio cuantitativo preliminar indicó que la desigualdad que más cortes introdujo fue *ADondeVaL*, introduciendo casi 6 veces la cantidad de cortes introducidos por *AlgúnOrigen* y *AlgúnDestino*, que fueron las siguientes familias en cantidad de cortes. Por último, las desigualdades *AlgúnOrdenIJ* y *TodoOrden3Puntos* lograron encontrar cortes válidos, pero sólo algunas decenas, cuando *ADondeVaL* encontró 1800 cortes.

Otra cosa a notar, es que de estas pocas decenas de cortes que encontraron las últimas dos familias, más del 90 % de estos fueron encontrados en el caso donde *alfa* es 1, o sea en el caso TSP.

Dado que las familias *AlgúnOrdenIJ* y *TodoOrden3Puntos* encontraron cortes casi solamente para el TSP, y en general este caso no presenta un desafio para el algoritmo que ya resuelve estas instancias en pocos segundos, se decidió realizar la experimentación sobre los tiempos de ejecución del algoritmo con las siguientes combinaciones de igualdades y desigualdades

- Sólo las igualdades cuadráticas. Ya presentado anteriormente, se incluyen para realizar la comparación.
- Igualdades cuadráticas con todas las desigualdades.
- Igualdades cuadráticas y las desigualdades, pero sin *AlgúnOrdenIJ* y *TodoOrden3Puntos*.

Nuevamente se presentan los tiempos de ejecución utilizados por cada combinación para resolver las mismas 6 instancias randoms ya utilizadas, con los mismos pares de parámetros. En los casos donde se consume la media hora de ejecución máxima, se reporta el gap porcentual obtenido.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	0.06	0.08	0.18	0.14	1.9
CI+TodasDes	0.06	0.08	0.18	0.14	1.84
CI+AlgunasDes	0.06	0.08	0.18	0.14	1.81

Cuadro 5.8: Tiempos para Instancia Random 1

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	1.68	404.33	(1.98 %)	826.78	893.79
CI+TodasDes	1.67	338.48	620.99	80.63	696.83
CI+AlgunasDes	1.67	330.62	605.22	78.72	678.42

Cuadro 5.9: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	0.29	6.36	45.12	365.96	251.19
CI+TodasDes	0.29	5.07	47.06	599.57	160.08
CI+AlgunasDes	0.29	4.93	45.80	578.70	155.05

Cuadro 5.10: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	0.03	0.04	0.04	0.05	0.23
CI+TodasDes	0.03	0.04	0.04	0.05	0.12
CI+AlgunasDes	0.03	0.04	0.04	0.05	0.12

Cuadro 5.11: Tiempos para Instancia Random 4

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	2.4	20.92	1440.5	570.07	1497.13
CI+TodasDes	3.06	44.25	1444.37	353.99	1069.39
CI+AlgunasDes	2.87	41.86	1513.37	343.46	1050.27

Cuadro 5.12: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI	5.2	(3.27 %)	(12.37 %)	(14.65 %)	(19.91 %)
CI+TodasDes	12.33	(3.07 %)	(13.42 %)	(17.16 %)	(21.75 %)
CI+AlgunasDes	11.25	(1.80 %)	(13.25 %)	(17,16 %)	(21.75 %)

Cuadro 5.13: Tiempos para Instancia Random 6

De los resultados presentados anteriormente se despreden varios puntos interesantes a analizar.

En primer lugar se hace mención a la diferencia entre incluir o no a las familias *TodoOrden3Puntos* y *TodoOrdenIJ*. Lo que se observa en casi todas las tablas presentadas es que la diferencia entre usar estas familias o no casi siempre produce el mismo efecto que es la disminución en una cantidad pequeña de segundos en los tiempos de ejecución.

Esta disminución en los tiempos, que no parece considerable, indica dos cosas. Por un lado indica que en los casos que no se utilizan estas familias, no hay mejoras considerables respecto de los otros casos, por lo que se nota que los tiempos de ejecución del algoritmo no tienen su mayor porcentaje en la resolución de los problemas de separación, porque sino la combinación sin estas familias debería verse beneficiada. Por otro lado, que los resultados sean coherentes entre todas las tablas indican que realmente estas familias no tienen un fuerte impacto práctico por lo que su inclusión no tiene una sólida fundamentación.

Luego, dado que la inclusión de estas familias no genera una mejora, si no que por el contrario suele devenir en una ligera desmejoría, se decide no incluirlas en el algoritmo.

Por otro lado, se puede ver que los comportamientos de las desigualdades incluidas se puede caracterizar dependiendo del tamaño de la instancia que se está tratando.

Las instancias pequeñas como la 1 y la 4, que poseen 16 y 12 nodos respectivamente, se observa que ya no aportan mucho a la discusión porque, desde las inclusión de las igualdades estudiadas anteriormente, todos los casos propuestos, inclusive el puramente angular, son resueltos hasta la optimalidad en cuestión de pocos segundos. En estas instancias es muy baja la variabilidad entre los resultados de las diferentes combinaciones estudiadas.

En una segunda clasificación se encuentran las instancias grandes, grupo que solo contiene a las instancia 6 con 46 nodos. Se observa que para esta instancia casi todas las combinaciones de parámetros sigue siendo un desafio no resuelto. Incluso se puede observar que en varios casos la inclusión de los planos de cortes empeoró el rendimiento del algoritmo, muy problablemente debido al overhead que generan los algoritmos de separación utilizados, que si bien son todos polinomiales, tienen ordenes de complejidad que pueden afectar al rendimiento del algoritmo con la cantidad de nodos que se está trabajando en esta instancia.

En una última clasificación se encuentran las instancias medianas, que resultaron las más interesantes en este punto, porque son las instancias en las que se pueden visualizar claras diferencias entre usar o no usar planos de corte. Las instancias que entran en esta clasificación son la 2, 3 y 5 con 35, 25 y 31 nodos respectivamente. Son instancias en el orden del doble de nodos de las chicas, pero con menos nodos respecto de las grandes, que marcan la barrera entre lo resoluble y lo no resoluble, en el tiempo máximo permitido de ejecución.

En estas instancias se observa que, en la mayoría de los casos, el uso de planos de corte es beneficioso en términos de tiempo de ejecución, sobretodo cuando el problema se parece más al puramente angular.

En la instancia número 2, se puede ver una mejoría en casi todas las combinaciones de parámetros. En el caso puramente angular con igualdades cuadráticas y las nuevas desigualdades, hay una mejoría considerable de casi 25 % de tiempo de ejecución.

Los resultados más notorios para la instancia 2, se dan en alfa 0.25 y 0.5 para las igualdades cuadráticas con las nuevas desigualdades.

En alfa igual a 0.5 se observa como se pasa de no haber resuelto el problema en la media hora fijada, reportando un gap de 1.98 %, a resolver la instancia en casi un tercio del tiempo máximo fijado.

El caso de mejoría más abrupto se observa en alfa igual a 0.25 en donde el tiempo de resolución con planos de corte bajo a un 10 % del tiempo de resolución original.

En la instancia número 3 también se observan mejorías en los casos con alfa pequeños, con considerables disminuciones porcentuales en los tiempos de ejecución, con algunos casos llegando a más del 50 %. En contraposición, esta instancia presenta un caso, alfa igual a 0.25, que se vió perjudicado por la utilización de planos de corte.

Por último, en la instancia número 5, también se observan mejoras consistentes en casi todos los casos donde alfa es menor o igual a 0.5.

Como observación general de todas estas instancias medianas, se puede ver que los planos de corte influyeron bastante en los tiempos de ejecución, sobre todo cuando el problema se parece más al puramente angular, lo cual es alentador ya que a priori estos eran los casos más desafiantes para el algoritmo.

Desigualdades *Comb* Dado que el TSP es un problema muy estudiado en la literatura, existen muchas familias para las que está demostrado que definen facetas del poliedro para este problema [12] [9] [10]. Una idea lógica para el problema del AngTSP es reutilizar estas desigualdades conocidas en la literatura para analizar como se comportan al insertarlas en esta generalización del TSP.

En este caso se utilizaron las desiguadaldes *Comb*, que muestran ser desigualdades muy importantes para el TSP, encontrándose implementadas en casi toda aproximación interesante al TSP.

Formulación e interpretación Las desigualdades *Comb* o *peine* se basan en tomar el grafo original de puntos a visitar y dividir los puntos en un conjunto H, denominado *handle* o *mango*, y s conjuntos T, denominados *teeth* o *dientes*. Para que la división sea válida, se deben cumplir las siguientes restricciones

$$T_i \cap T_j = \emptyset \qquad \forall i, j = 1 \dots s, i \neq j \tag{5.18}$$

Es decir, los dientes no tienen intersección.

$$\mid H \cap T_i \mid \geq 1 \qquad \forall i = 1 \dots s \tag{5.19}$$

Lo que quiere decir que los dientes están unidos al mango.

$$\mid T_i \setminus H \mid \geq 1 \qquad \forall i = 1 \dots s \tag{5.20}$$

Lo que indica que los dientes son algo más que la intersección con el mango.

Si se cumplen las anteriores restricciones, y s es un número impar mayor o igual a 3, entonces vale la siguiente desigualdad

$$\sum_{i,j=1,i,j\in H}^{n} x_{ij} + \sum_{t=1}^{s} \sum_{i,j=1,i,j\in T_t}^{n} x_{ij} \leq \mid H \mid + \sum_{t=1}^{n} (\mid T_t \mid -1) - (s+1)/2 \tag{5.21}$$

Dimensión Las desigualdades *Comb* están probadas como generadoras de facetas para el TSP tanto en su versión simétrica como asimétrica. Para este problema en particular se las incluyó debido a esta propiedad, pero no se experimentó con la dimensión en el problema actual.

Separación El algoritmo de separación para esta familia de desigualdades no resulta nada trivial. En su versión más general este algoritmo no es polinomial, aunque existen algoritmos polinomiales para aproximar el problema o para separar subfamilias de desigualdades dentro de las *Comb*, como se presenta en [14]. Para este trabajo se utilizó una implementación del algoritmo de separación utilizada en [16]. Para dicho artículo se utilizaron varias rutinas que se encuentran conglomeradas en el paquete CVRPSEP presentadas en [15].

Es importante notar que el paquete utilizado no fue desarrollado para el TSP, sino que fue utilizado para el CVRP(*Capacitated Vehicle Routing Problem*). Lo que se tuvo que hacer fue realizar la traducción del problema de este trabajo al CVRP, y adecuarlo a las condiciones necesarias. Lo más importante a notar es que el AngTSP formulado en este trabajo es a priori asimétrico ya que existen los ejes de ida y de vuelta entre dos nodos, aunque todas las instancias utilizadas hasta ahora son simétricas por ser euclideas; y la separación provista en CVRPSEP es para el problema simétrico, por lo que el paquete nos entregará desigualdades válidas, pero no todas las que podrían estar presentes en la versión asimétrica.

A continuación se presentan los resultados de incluir las desigualdades *Comb* en el algoritmo. Se utilizaron las mismas 6 instancias random utilizadas hasta el momento, aunque no se presentan los resultados para las instancias 1 y 4 dado que, gracias a su tamaño, ya son resueltas rápidamente con cualquier combinación de desigualdades.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	1.67	330.62	605.22	78.72	678.42
CI+Comb	1.66	468.67	652.58	78.54	770.97

Cuadro 5.14: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	0.29	4.93	45.80	578.70	155.05
CI+Comb	0.29	3.16	42.97	739.04	125.64

Cuadro 5.15: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	2.87	41.86	1513.37	343.46	1050.27
CI+Comb	2.31	61.98	1587.74	601.35	1234.25

Cuadro 5.16: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	11.25	(1.80 %)	(13.25 %)	(17.16 %)	(21.75 %)
CI+Comb	5.46	(5.26 %)	(16.25 %)	(18.10 %)	(20.22 %)

Cuadro 5.17: Tiempos para Instancia Random 6

Como se observa en los resultados presentados, la inclusión de las desigualdades *Comb* no sólo no generó un beneficio en casi ningún caso, si no que en la mayoría perjudicó los tiempos de ejecución. Al parecer, para la mayoría de los casos, el tiempo utilizado por el algoritmo de separación no logra un buen balance con la retribución que generan los cortes encontrados.

Lo único para notar a favor de estas desigualdades es como se redujo el tiempo para el caso de TSP en la instancia número 6, que es la de más nodos, esto parecería ser un indicio de que realmente las desigualdades sirven para el caso donde el problema es el TSP, pero no se adecúan bien al problema angular.

5.3. Eliminación de desigualdades

En todas las secciones anteriores el objetivo era tomar el modelo original y fortacelerlo mediante la inclusión de diferentes igualdades o desigualdades que eliminasen puntos fraccionarios de la región de interés. En este caso la mejora es a la inversa, la idea ahora es sacarle restricciones al modelo que antes eran necesarias para que el mismo fuese correcto y que ahora, gracias a la inclusión de todo lo presentado en las anteriores secciones, no son necesarias.

En este caso la mejora provino de analizar la desigualdad 3.36 que se encontraba originalmente en el modelo.

$$y_{ijk} \geq x_{ij} + x_{jk} - 1, \qquad i, j, k = 1, \ldots, n$$

Esta desigualdad establece que si dos ejes estan elegidos en la solución, entonces se está utilizando el ángulo que forman estos dos ejes.

Esta restricción en un primer momento era necesaria para el modelo, porque era la única restricción que obligaba a ciertas y a tomar un valor positivo, sin esta restricción la solución óptima siempre hubiese sido fijar todas las variables y en 0, arrojando una solución ficticia en cuanto a la realidad que se está intentando modelar.

Luego de las inclusiones en el modelo de las igualdades estudiadas, esta desigualdad pasa a ser innecesaria para la correción del mismo. Se puede demostrar que esta desigualdad pasa a ser superflua para los puntos enteros.

Lo que se quiere ver es que dadas las igualdades estudiadas, no puede suceder el hecho de que se utilicen dos aristas y no se esté utilizando el ángulo conformado por ellas.

En la siguiente demostración, siempre se hace mención a indices diferentes, dado que son las variables que están definidas para el problema.

Proposición. Si $x_{ij} = 1$ y $x_{jk} = 1$, entonces $y_{ijk} = 1$.

Demostración. Sean x_{ij} y x_{jk} dos variables de aristas tal que

$$x_{ij} = 1$$

$$x_{jk} = 1$$

para ciertos i, j y k.

Luego, por la igualdad 5.6, tiene que existir exactamente una variable y_{ija} tal que

$$y_{ija} = 1$$

para algún a.

Supongase que a es distinto de k. Entonces, por la relación 5.7, se cumple

$$x_{ja} = 1$$

Entonces se tienen a las variables x_{jk} y x_{ja} en 1 con a distinto de k. Esto contradice la relación 3.2 generando un absurdo que provino de suponer que a era distinto de k.

Luego, si se tienen en 1 las variables x_{ij} y x_{jk}, en cualquier solución que cumpla las restricciones, se tendrá en 1 la variable y_{ijk} como se buscaba originalmente.

□

Esto demuestra que no se necesita la inclusión de la relación 3.36 en el modelo para que el mismo sea correcto. Este resultado es alentador ya que estas desigualdades son las que hacían que la cantidad de restricciones del modelo original perteneciesen a $\mathcal{O}(n^3)$, ya que las de rompimiento de subtours se agregan bajo demanda.

Al excluir estas desigualdades del modelo, se obtiene una cantidad de restricciones perteneciente a $\mathcal{O}(n^2)$, por lo que se espera reducir los tiempos de ejecución totales, al reducir el tiempo necesitado para resolver cada una de las relajaciones lineales del árbol recorrido.

A continuación, se presentan los tiempos de ejecución conseguidos al eliminar dichas desigualdades del modelo. Nuevamente se utilizan las mismas 6 instancias random, con las mismas combinaciones de parámetros y media hora de tiempo de ejecución máximo. Las filas con sufijo *AlgunaDes*, son los modelos que se venían utilizando hasta el momento con algunas desigualdades válidas, mientras que las filas con sufijo *SinCúbicas*, son los modelos sin las desigualdades recientemente analizadas.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	0.06	0.08	0.18	0.14	1.81
CI+SinCubicas	0.03	0.04	0.06	0.06	0.52

Cuadro 5.18: Tiempos para Instancia Random 1

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	1.67	330.62	605.22	78.72	678.42
CI+SinCubicas	0.71	144.05	150.92	57.46	201.05

Cuadro 5.19: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	0.29	4.93	45.80	578.70	155.05
CI+SinCubicas	0.17	1.24	8.20	120.19	55.94

Cuadro 5.20: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	0.03	0.04	0.04	0.05	0.12
CI+SinCubicas	0.02	0.02	0.02	0.03	0.17

Cuadro 5.21: Tiempos para Instancia Random 4

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	2.87	41.86	1513.37	343.46	1050.27
CI+SinCubicas	1.57	33.49	379.24	576.35	126.29

Cuadro 5.22: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+AlgunasDes	11.25	(1.80 %)	(13.25 %)	(17.16 %)	(21.75 %)
CI+SinCubicas	9.44	1284.65	(7.01 %)	(9.03 %)	(7.40 %)

Cuadro 5.23: Tiempos para Instancia Random 6

De las anteriores tablas se puede ver que, como se esperaba, la eliminación de las desigualdades de orden cúbico resultó de gran ayuda para los tiempos de ejecución, presentando mejoras en casi todos los casos. Los tiempos de ejecución se redujeron en el orden del 50 % en muchos de los casos, llegando a picos de casi 90 %.

Es por todo lo presentado anteriormente que se concluye que la eliminación de las restricciones superfluas produce relajaciones lineales más livianas para resolver, y por lo tanto los algoritmos deberían ser capaces de explorar una mayor cantidad de nodos. De aquí en más, las desigualdades en cuestión no serán utilizadas en el modelo.

Capítulo 6

Heurísticas Primales, generación y recorrido del árbol

6.1. Heurísticas Primales

Otra de las herramientas útiles para los algoritmos de tipo branch-and-cut es la utilización de heurísticas primales. A diferencia de las iniciales, las heurísticas primales tienen por objetivo encontrar buenas soluciones enteras (cotas primales) no al comenzar el algoritmo, sino al tener más información. En particular, estas heurísticas tratan de encontrar una buena solución entera en cada nodo del árbol de branch-and-cut utilizando la información provista por el nodo actual, lo que en general se traduce en la resolución de la relajación lineal del nodo.

La resolución de la relajación lineal en un nodo puede otorgar información importante para generar una solución entera de calidad. Por ejemplo, si una variable binaria tiene un valor muy cercano a 1 en la solución fraccionaria, es lógico creer que en la solución óptima entera esta variable estará en 1 y no en 0.

Otra diferencia usual con las heurísticas iniciales es en cuanto a los tiempos de ejecución. Las heurísticas primales, a priori, se ejecutan en cada uno de los nodos del branch-and-cut, por lo que se deben utilizar algoritmos que no consuman mucho tiempo de corrida.

A continuación se presentan tres heurísticas primales analizadas en el presente trabajo.

6.1.1. Xgreedy

La primera heurística primal desarrollada es trivial tomando en cuenta la idea mencionada de hacer parecer la solución entera a la solución fraccionaria en la medida de lo posible.

La idea es tomar solamente la información respectiva a los ejes de la solución fraccionaria, es decir los valores de las variables x_{ij} en el óptimo de la relajación lineal, y en base a esto ir armando un ciclo hamiltoniano, que se traducirá de manera directa en los valores de una solución entera factible.

Lo primero que se hace es buscar al eje que tenga el mayor valor de la variable correspondiente en la relajación lineal, este eje será utilizado en la solución y, por lo tanto, se procede a colocar los dos nodos en cuestión como los primeros dos nodos del ciclo.

Luego, mientras haya algún nodo que no se encuentre en el ciclo, se insertará a la solución el eje que mayor valor tenga, mirando solamente los ejes entre el último nodo agregado al ciclo y todos los nodos que todavía no se encuentran en el mismo.

De esta forma, se cuenta con un prodecimiento goloso que busca formar un ciclo utilizando la información de la relajación lineal.

A continuación se muestra el pseudocódigo de dicha heurística

function XGREEDY

```
    resultado ← []
    mejorEje ← eje con mayor valor en relajacion lineal
    agregar cola de mejorEje a la cola de resultado
    agregar cabeza de mejorEje a la cola de resultado
    while resultado no contenga todos los nodos do
        actual ← cabeza del eje con mayor valor entre los ejes con el ultimo nodo de resultado como cola
        agregar actual a la cola de resultado
    end while
    devolver resultado
end function
```

Complejidad Para analizar la complejidad temporal del algoritmo, se analiza cada paso por separado. En primer lugar, se utiliza un arreglo para indicar si cada nodo ya fue utilizado o no en el ciclo, el seteo de dicho arreglo toma tiempo lineal en la cantidad de puntos. Luego, se buscan los dos primeros nodos del ciclo, buscando la variable de eje con mayor valor. Para esto simplemente hay que recorrer todos los pares de nodos, por lo que este paso tiene una complejidad temporal cuadrática.

Luego, se va buscando el próximo nodo a agregar en tiempo lineal ya que se recorren potencialmente todos los nodos y se realizan operaciones con complejidad constante. Este paso se realiza tantas veces como nodos se tenga, por lo que este paso también tiene complejidad cuadrática.

Por último, algo para destacar que es meramente implementativo pero impacta en la complejidad, es que en CPLEX la solución factible encontrada por una heurística primal se debe entregar al framework explicitando el valor de cada una de las variables del modelo.

De esta forma, si bien todos los pasos anteriores tienen complejidad cuadrática o menor, el algoritmo total pasa a tener una complejidad temporal perteneciente a $\mathcal{O}(n^3)$ cuando se sobrescribe la solución factible. El paso de escribir una solución factible solo sucede cuando el algoritmo encuentra una mejor que la que ya se tiene, por lo que en la mayoría de los nodos el algoritmo tomará tiempo cuadrático.

6.1.2. Ygreedy

La segunda heurística primal analizada es completamente análoga a la presentada recientemente, pero en vez de utilizar los valores de las variables x de la relajación para ir armando el ciclo, utiliza los valores de las variables y. Es decir, primero coloca en el ciclo los 3 nodos correspondientes a la varible y_{ijk} con mayor valor en el óptimo fraccionario. Luego, se busca la variable y_{ijk} con mayor valor, pero solamente mirando las variables que tienen como i y j a los dos ultimos nodos del ciclo formado hasta el momento, y como k a todos los nodos que todavía no se encuentran dentro del ciclo. Este último paso se repite hasta tener todos los nodos dentro del ciclo.

A continuación, se presenta el pseudocódigo de dicha heurística.

```
function YGREEDY
    resultado ← []
    mejorAngulo ← angulo con mayor valor en relajacion lineal
    agregar primer nodo de mejorAngulo a la cola de resultado
    agregar segundo nodo de mejorAngulo a la cola de resultado
    agregar tercer nodo de mejorAngulo a la cola de resultado
    while resultado no contenga todos los nodos do
        mejorAngulo ← angulo con mayor valor entre los que usan los dos ultimos nodos del ciclo
        agregar tercer nodo de mejorAngulo a la cola de resultado
    end while
    devolver resultado
```

end function

Complejidad El análisis de complejidad temporal es completamente análogo al realizado en la heurística primal *Xgreedy*, con la excepción de que la elección de los primeros puntos del ciclo ya toma tiempo cúbico porque se busca el ángulo de mayor valor en la solución fraccionaria. Por lo que, indistintamente de si el algoritmo mejora o no la solución óptima actual, la complejidad temporal pertenece a $\mathcal{O}(n^3)$.

6.1.3. MSTgreedy

La tercera heurística primal utilizada en este trabajo es derivada de lo que en la literatura se conoce como heúristica del árbol generador mínimo [3].

Como su nombre lo indica, la idea es armar un circuito hamiltoniano a partir del árbol generador mínimo del grafo, pero en este caso la noción de peso en un eje no estará dada por la distancia entre los nodos, sino que se quiere utilizar el valor de cada eje en la solución de la relajación del nodo, y para seguir refiriéndose al árbol generador mínimo lo que se utilizará serán estos pesos pero en negativo, ya que en realidad se quieren priorizar los valores cercanos a 1 en la relajación.

La heurística consiste de los siguiente pasos, que conforman una noción de pseudocódigo.

1. Se ordenan los ejes de menor a mayor por el valor negativo de los pesos de cada eje en la solución de la relajación lineal.
2. Con los ejes ordenados correctamente, se procede a generar el árbol generador mínimo, bajo esta noción de peso, mediante el algoritmo de Kruskal.
3. Se duplican los ejes entre cada par de nodos del árbol y se busca un circuito euleriano en el grafo.
4. Se genera el circuito hamiltoniano deseado recorriendo el euleriano encontrado en el paso anterior y salteando los nodos repetidos.

Cabe notar que si bien en la mayoría de la literatura se presentan los dos últimos pasos de la heurística de la misma manera que fue hecha en esta explicación, en la práctica se suele hacer un sólo paso que resume los dos, que es ejecutar una *búsqueda en profundidad* o *Depth First Search* desde algún nodo distinguido.

Complejidad Al igual que en las heurísticas anteriores, para analizar la complejidad temporal del algoritmo se analizará cada paso por separado.

En primer lugar se inicializan arreglos auxiliares necesarios para el algoritmo para saber, por ejemplo, si el nodo fue visitado en el *DFS* o cuál es el padre de cada nodo para realizar el árbol generador mínimo. Estas inicializaciones tienen una complejidad temporal lineal en la cantidad de nodos.

Luego, para ejecutar el algoritmo de Kruskal primero se deben tener los ejes ordenados, en este caso por el criterio propuesto en esta heurística. Dicho paso ordena una cantidad cuadrática de ejes, ya que el grafo es completo, por lo que este paso tiene una complejidad temporal perteneciente a $\mathcal{O}(n^2 log(n^2))$, o análogamente $\mathcal{O}(n^2 log(n))$.

En el siguiente paso se procede a armar el árbol generador mínimo lo cual, utilizando la técnica de *union-find con path-compression*, tiene una complejidad temporal pertenciente a $\mathcal{O}(n^2 ack^{-1}(n))$. Donde ack^{-1} se refiere a la inversa de la función de Ackermann.

Dado que el armado del árbol tiene una complejidad temporal que está incluida en la que toma ordenar los ejes, se concluye que todo el algoritmo de Kruskal tiene una complejidad temporal perteneciente a $\mathcal{O}(n^2 log(n))$.

Por último, el *DFS* tiene una complejidad temporal lineal en el tamaño del grafo, lo cual en un grafo completo es análogo a decir que pertenece a $\mathcal{O}(n^2)$.

Al igual que en las dos heurísticas anteriores, es importante destacar que si se encuentra una solución factible mejor que la actual, entonces la sobrescritura de la misma toma tiempo cúbico para indicar el valor de cada una de las variables del modelo.

Búsqueda local Análogamente a lo hecho con las heurísticas iniciales, es razonable ejecutar búsquedas locales para mejorar las soluciones entregadas por las heurísticas primales. Sin embargo, ejecutar una búsqueda local cada vez que se ejecuta una heurística primal problablemente resulte perjudicial en cuanto a tiempos de ejecución.

Es por lo dicho anteriormente que en el presente análisis se observaron tres opciones diferentes.

- Cada heurística primal de las presentadas, sin realizar una búsqueda local posterior.
- Realizar una búsqueda local 3-opt cada vez que se ejecuta una heurística primal.
- Realizar una búsqueda local 3-opt luego de las heurísticas primales, pero solamente en los nodos superiores del árbol de branch-and-cut.

La tercera opción es un compromiso entre las dos primeras, para lograr tener las bondades de la búsqueda 3-opt, pero sin ejecutarla demasiadas veces. La idea es que el algoritmo encuentre soluciones factibles de calidad al comienzo del mismo para tener cotas más ajustadas para todo el resto del árbol.

Experimentación A continuación se presentan los tiempos de ejecución obtenidos para todas las combinaciones mencionadas.

Las pruebas se realizaron sobre las mismas 6 instancias que se vienen analizando a lo largo del trabajo.

Nuevamente se excluyen los resultados de las instancias 1 y 4 dado que, gracias a su tamaño y las mejoras anteriores en el algoritmo, ya son resultas en tiempos indistinguibles para las diferentes combinaciones. También se excluyen todas las combinaciones en donde se realiza la búsqueda 3-opt ya que, como se esperaba, sus tiempos de ejecución fueron notoriamente más altos debido al overhead lógico producido al correr la metaheurística.

Al igual que en los resultados previos, cuando el algoritmo consumió la media hora de tiempo límite fijada, se reporta el gap obtenido.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+SinPrimal	0.71	144.05	150.92	57.46	201.05
CI+Xgreedy	0.71	231.55	287.94	124.60	294.91
CI+Xgreedy3optarriba	1.29	103.53	285.54	157.27	329.50
CI+Ygreedy	0.71	234.92	244.99	124.95	296.54
CI+Ygreedy3optarriba	1.44	124.98	172.27	159.23	466.54
CI+MSTgreedy	0.71	222.74	350.60	124.87	294.96
CI+MSTgreedy3optarriba	1.24	275.83	358.10	157.15	459.29

Cuadro 6.1: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+SinPrimal	0.17	1.24	8.20	120.19	55.94
CI+Xgreedy	0.17	2.00	9.79	135.50	82.00
CI+Xgreedy3optarriba	0.17	1.50	10.21	113.07	73.97
CI+Ygreedy	0.17	1.42	9.73	136.45	72.00
CI+Ygreedy3optarriba	0.17	1.30	10.46	64.10	84.68
CI+MSTgreedy	0.17	1.82	9.96	135.75	75.62
CI+MSTgreedy3optarriba	0.17	1.38	10.34	169.33	87.29

Cuadro 6.2: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+SinPrimal	1.57	33.49	379.24	576.35	126.29
CI+Xgreedy	1.95	35.53	519.07	713.31	144.66
CI+Xgreedy3optarriba	3.01	41.78	338.56	720.80	112.84
CI+Ygreedy	1.93	34.33	521.30	718.03	145.33
CI+Ygreedy3optarriba	3.58	47.07	417.86	720.73	147.71
CI+MSTgreedy	1.94	34.17	337.62	718.14	144.56
CI+MSTgreedy3optarriba	2.87	47.11	266.91	714.47	88.60

Cuadro 6.3: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+SinPrimal	9.44	1284.65	(7.01 %)	(9.03 %)	(7.40 %)
CI+Xgreedy	10.98	(1.39 %)	(8.22 %)	(13.37 %)	(12.74 %)
CI+Xgreedy3optarriba	21.66	(1.37 %)	(5.71 %)	(10.54 %)	(10.70 %)
CI+Ygreedy	11.28	(1.65 %)	(8.22 %)	(8.27 %)	(8.85 %)
CI+Ygreedy3optarriba	26.60	(2.48 %)	(5.22 %)	(10.66 %)	(9.06 %)
CI+MSTgreedy	11.11	(2.36 %)	(8.20 %)	(9.14 %)	(7.89 %)
CI+MSTgreedy3optarriba	23.83	(2.42 %)	(7.57 %)	(9.31 %)	(11.91 %)

Cuadro 6.4: Tiempos para Instancia Random 6

De las tablas presentadas anteriormente se observa que no hay tendencias claras para concluir que alguna de las combinaciones presentadas es mejor que las demás. De hecho, en varias de las columnas lo más concluyente es que la combinación sin ninguna heurística primal es la mejor combinación.

Dado que este no era el resultado esperado, ya que las buenas propiedas de las heurísticas primales son ampliamente fundamentadas en la literatura, se realizó una observación más minuciosa sobre las corridas, mirando no solamente los tiempos de ejecución sino también la cantidad de nodos explorados y la mejor solución entera encontrada en el caso de que el algoritmo no haya terminado en el tiempo límite.

Se realizaron las siguientes observaciones importantes para explicar el comportamiento visto en las tablas:

En primer lugar, cabe destacar que CPLEX trabaja por defecto con heúristicas primales. Si bien

estas heurísticas son generales y no utilizan información particular del problema, en las corridas se puede observar que varias veces estas heurísticas logran encontrar soluciones enteras mejores que las que tenía hasta el momento. Es decir, que la combinación que en las tablas aparece con la leyenda *SinPrimal*, en realidad sí tiene heurísticas primales aplicadas, que son las que provee el paquete CPLEX.

En segundo lugar, la observación más importante es en cuanto a la calidad de las heurísticas iniciales dado los tamaños de instancia que se está manejando. Lo que se observa es que las heurísticas iniciales utilizadas encuentran soluciones enteras que, o son óptimas, o están muy cerca del óptimo. Es razonable que entonces la ejecución de las heurísticas primales sólo resulten en un overhead temporal, no pudiendo hallar mejores soluciones.

Para observar esto, se constrastó la performance de las combinaciones actuales, contra las mismas combinaciones pero sin las heurísticas iniciales. Al hacer esto, en la mayoría de los casos se pudo advertir la utilidad de las heurísticas primales satisfactoriamente.

Por último, se destaca que las heurísticas iniciales arrojan tan buenas soluciones, debido a que el tamaño de las instancias que se están manejando permiten la exploración de un buen porcentaje de las soluciones factibles en poco tiempo. Sin embargo, es posible que al aumentar la cantidad de nodos de una instancia, las heurísticas iniciales ya no encuentren soluciones tan cercanas al óptimo, y en ese caso sí es probable que se pueda notar la utilidad de las heurísticas primales presentadas.

Para esto, lo que se hizo fue realizar la ejecución sobre una instancia de 76 nodos. Si bien es completamente esperable que la resolución de la instancia no termine en la media hora fijada, excepto para el caso de alfa igual a 1, lo que se quiere ver es la calidad de las soluciones factibles encontradas para ver si las heurísticas primales realizan un salto de calidad respecto de las iniciales.

A continuación se presentan los tiempos de ejecución y los gaps obtenidos, en el caso de que se utilice la media hora de tiempo límite, para la instancia Eil76 de la TSPLIB. Todas las combinaciones tienen habilitadas las heurísticas iniciales y lo que se varió fue la primal.

Heurística \ alfa	1.0	0.75	0.5	0.25	0.0
PrimalCPLEXDeshabilitada	65.53	(15.27 %)	(21.01 %)	(22.70 %)	(24.06 %)
PrimalCPLEXHabilitada	43.20	(15.27 %)	(21.01 %)	(22.70 %)	(24.06 %)
Xgreedy3optarriba	37.40	(10.13 %)	(17.12 %)	(19.91 %)	(24.06 %)
Ygreedy3optarriba	28.44	(12.52 %)	(17.47 %)	(21.74 %)	(24.06 %)
MSTgreedy3optarriba	75.13	(10.97 %)	(16.55 %)	(19.49 %)	(24.06 %)

Cuadro 6.5: Instancia Eil76 de TSPLIB

Nuevamente con este experimento se pudo observar lo esperado en la mayoría de las combinaciones. Al utilizar instancias más grandes las heurísticas iniciales ya no son tan efectivas y la inclusión de las heurísticas primales resulta conveniente en varios de los casos presentados llegando a reducir el gap en porcentajes no despreciables.

Los casos en los que no se denota mejora son los correspondientes al problema puramente angular, para analizar esta situación se revisó cada corrida en particular. Se pudo observar que ninguna de las combinaciones es capaz de encontrar una solución entera mejor que la arrojada por la heurística inicial, por lo que está la posibilidad de que esta sea la óptima y por eso no se vean diferencias entre las opciones para las primales.

Por otro lado, hay un caso en particular que no dió lo esperado, que es el caso TSP para la heurística *MSTgreedy3optarriba*. Este resultado no es el esperado ya que la ejecución tomó más tiempo que la opción sin ningún tipo de heurística. Al observar las corridas de estos dos casos particulares, se ve que la opción sin heurística encuentra el óptimo rápidamente al encontrarse con una solución entera de una relajación lineal, mientras que la opcion de árbol generador mínimo encuentra siete soluciones factibles que mejoran a la precedente antes de llegar al óptimo.

Más allá de los casos particulares explicados, parecería notarse un beneficio en la inclusión de las heurísticas primales desarrolladas al aumentar la cantidad de nodos.

6.2. Generación y recorrido del árbol

El último aspecto algorítmico a analizar en este trabajo es el relacionado a la forma en la que se genera y se recorre el árbol de branch-and-cut.

Al ejecutar un algoritmo de branch-and-cut se genera un árbol con los diferentes subproblemas a resolver, y se lo va recorriendo en cierto orden para ir resolviendo los diferentes nodos creados hasta que se consiga la solución óptima y el certificado de su optimalidad. Esta tarea encuadra dos decisiones, por un lado se debe tomar una decisión para ver como se irá creando el árbol, como nacerán los nuevos subproblemas; por el otro, luego de haber fijado la elección para el paso recien mencionado, se debe elegir cual va a ser el próximo nodo o subproblema a resolver.

Estas decisiones pueden tener alto impacto en la calidad del algoritmo, ya que dependiendo el árbol que se crea y como se lo recorra se puede encontrar la solución muy rápidamente o se puede resolver muchos subproblemas inútiles antes de llegar a la solución óptima.

Se analizarán entonces las opciones para la creación de nuevos nodos, y las opciones para la elección del próximo nodo a resolver.

6.2.1. Branching

El proceso de branching es justamente el que se ocupa de la ramificación del árbol. Es decir, cuando se resuelve la relajación lineal de un nodo y éste no se puede descartar, hay que tomar una decisión sobre qué subproblemas crear para este nodo.

Existen diferentes estrategias de branching, algunas son desde un enfoque global, independizándose del problema, mientras que otras estrategias se suelen generar para cada problema en particular.

En este trabajo se analizaron 5 estrategias de branching diferentes. Por el lado general, el paquete CPLEX ofrece varias estrategias de branching para utilizar, en este caso se optó por analizar 3 de ellas. En primer lugar se analizó la estrategia automática de CPLEX, ya que es la que estuvo implícitamente presente en las secciones anteriores. Luego, también se analizaron las estrategias *strong branching* y *pseudo-costos reducidos*, ya que son dos de las estrategias ofrecidas que son recomendadas en la literatura [1].

Por otro lado, también se analizaron dos estrategias particulares para este problema. Estas estrategias están basadas en el hecho de otorgarle a cada variable del modelo una prioridad diferente para que se realice el branching por dicha variable. De esta forma, utilizando los significados que tienen las variables del modelo en el problema real, se puede esperar ciertos resultados al otorgar determinadas prioridades.

En el problema tratado no existe distinción entre los nodos o entre los ejes, es por esto que las 2 estrategias particulares que se observaron son simplemente dar mayor prioridad a las variables de eje, las variables x, por sobre las variables de ángulo, las y, y viceversa.

La observación de los tiempos de ejecución se realizó sobre las mismas 6 instancias que se utilizaron en las anteriores secciones. Nuevamente se omiten los resultados de las instancias 1 y 4 que ya no muestran diferencias entre las distintas opciones al estar resueltas en fracciones de segundo. Cuando el algoritmo no finalizó en la media hora de tiempo de ejecución prefijada, se reporta el gap alcanzado.

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+Default	0.71	143.67	155.18	58.16	203.09
CI+proriodadX	0.70	97.18	200.69	80.00	128.78
CI+proriodadY	0.68	195.85	397.71	275.39	479.94
CI+Pseudo	0.61	121.91	137.46	93.11	212.14
CI+Strong	0.70	57.30	141.32	94.69	140.57

Cuadro 6.6: Tiempos para Instancia Random 2

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+Default	0.17	1.23	8.40	122.76	56.59
CI+proriodadX	0.17	1.27	7.20	93.30	39.96
CI+proriodadY	0.17	2.66	11.42	136.91	113.23
CI+Pseudo	0.17	0.94	8.24	77.72	238.08
CI+Strong	0.17	1.05	16.75	50.87	55.13

Cuadro 6.7: Tiempos para Instancia Random 3

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+Default	1.59	33.88	382.27	584.23	127.25
CI+proriodadX	2.11	19.65	277.79	382.07	74.48
CI+proriodadY	1.61	23.30	973.95	809.19	491.34
CI+Pseudo	3.35	12.49	335.64	258.50	157.52
CI+Strong	1.54	6.07	324.51	221.71	61.80

Cuadro 6.8: Tiempos para Instancia Random 5

Modelo \ alfa	1.0	0.75	0.5	0.25	0.0
CI+Default	9.64	1296.90	(7.02 %)	(9.05 %)	(7.45 %)
CI+proriodadX	11.43	(0.58 %)	(7.96 %)	(9.86 %)	(7.36 %)
CI+proriodadY	19.14	(1.08 %)	(10.18 %)	(18.99 %)	(23.98 %)
CI+Pseudo	26.89	1645.21	(8.51 %)	(9.24 %)	(9.31 %)
CI+Strong	6.38	852.13	(6.53 %)	(7.89 %)	(7.83 %)

Cuadro 6.9: Tiempos para Instancia Random 6

De los resultados anteriores se desprenden varias observaciones para realizar la elección de una estrategia de branching adecuada.

En primer lugar, analizando las dos estrategias particulares para este problema, se puede ver que la opción en donde se le da prioridad a las variables de eje funciona mejor, en casi todos los casos, que la opción de prioridad a las variables de ángulo. De hecho se puede observar que la opción de prioridad a las x, funciona mejor que el branching automático en varios casos, mientras que la opción de prioridad a las y, tiene varios casos en donde funciona peor que la opción automática.

Esta comparación entre estas dos estrategias tiene una explicación en función de la estructura de los árboles que crea cada una de las opciones. Cuando se ramifica por una variable x, en una de las dos ramas seguro no se usa el eje por el que se ramificó, y en la otra rama seguro sí se lo usa. No utilizar un eje no restringe demasiado la elección de los otros ejes del ciclo, mientras que tener una rama en donde seguro se utiliza un eje en particular sí hace que ese subproblema sea bastante más restrictivo. Es por eso que se dice que el árbol está desbalanceado, en el sentido de que los dos subproblemas que quedan abiertos tienen restricciones diferentes que resultan en muchas más soluciones factibles para uno que para el otro.

Sin embargo, cuando se utiliza la ramificación por una variable de ángulo la diferencia entre los dos problemas es mucho mayor. En la ramificación que no se utiliza el ángulo, lo único que se está diciendo es que esos dos ejes no pueden ser utilizados en estrictamente ese orden; en cambio, en el subproblema donde si se utiliza el ángulo se restringe a la utilización de los dos ejes involucrados y en ese determinado orden. Por lo cual se produce un desbalance aún mayor entre los dos subproblemas que se crean al ramificar un nodo por una variable y.

En general, en un algoritmo branch-and-cut, se suelen obtener mejores resultados cuando el árbol es más balanceado. Esto es justamente lo que explica el mejor desempeño de la estrategia de prioridad a las x por sobre las y.

Luego, observando las estrategias ofrecidas por CPLEX, se puede ver que la opción de *strong branching* tiene un mejor rendimiento por sobre la opción de pseudocostos reducidos, así como por sobre la elección automática de CPLEX. Es por eso que para estos casos analizados, se elige a la estrategia de *strong branching* como la mejor entre las ofrecidas por el framework.

El último análisis sobre branching, es entre la estrategia de prioridad a las variables x y el *strong branching* de CPLEX, para poder elegir una sola estrategia a utilizar.

Cuando se observan estas dos opciones se pueden ver comportamientos cruzados, no quedando claro si algunas de las estrategias es mejor que la otra. La única pequeña diferencia se produce al observar las combinaciones de parámetros donde el problema es angular, es decir cuando el alfa es 0. En estos casos la estrategia que da prioridad a las variables x tiene un comportamiento ligeramente superior al *strong branching*, por lo que se elige dicha estrategia para ser utilizada en el algoritmo final ya que las instancias angulares son las que resultan más desafiantes.

6.2.2. Selección de nodo

La selección de nodo es la decisión de qué subproblema se resolverá primero. Con el proceso de branching que fue analizado recientemente, el árbol del branch-and-cut se va ramificando y van creándose nuevos nodos o subproblemas que se posicionan en una lista de nodos activos para ser resueltos. La selección de nodo es justamente elegir cuál va a ser el próximo nodo de la lista de activos que se va a resolver.

Este tipo de elección también puede tener un alto impacto en la calidad del algoritmo. Por ejemplo, si se va cada vez más profundo en el árbol, los subproblemas tienen varias variables ya fijadas, esto hace que sea más simple encontrar una solución factible al problema pero a la vez, dado que hay muchas variables fijadas, esta solución no tiene por qué ser la óptima global. Luego, para un problema donde es difícil encontrar una solución factible de calidad para tener una buena cota primal, problablemente la mejor opción sea recorrer el árbol en profundidad para tratar de hallar más rapido soluciones factibles.

El paquete CPLEX ofrece cuatro opciones diferentes para la selección de nodo. La opción *best bound*, que es la que se usa por defecto en CPLEX, por lo que fue la que implícitamente se utilizó hasta el momento; la opción de depth-first search o búsqueda en profundidad como se mencionó recientemente, y dos variedades de *best estimate search*.

Al igual que en las secciones anteriores, el análisis se basó en ejecutar las diferentes opciones sobre las mismas 6 instancias que se venía trabajando y con las mismas combinaciones de parámetros. En este caso no se presentan los resultados, ya que los mismos no proporcionan información

interesante de alguna estrategia por sobre las demás. Las observaciones más relevantes de las ejecuciones hechas son

- En la gran mayoría de los casos, la variabilidad porcentual en los tiempos de ejecución no es significativa entre la mejor opción y la siguiente.
- Si bien hay casos donde la opción de dfs es mejor que las demás, existen varios casos en donde esta opción queda última y con una performance notoriamente peor que las demás elecciones.
- Entre las dos alternativas de *best estimate search*, existe poca diferencia en la performance, siendo una de ellas ligeramente mejor que la otra.
- Entre la alternativa por defecto, *best bound*, y la mejor de las dos de *best estimate search*, las diferencias son muy pequeñas.

Es por todo lo visto en los resultados que se elige como opción para la selección de nodo a la estrategia *best bound* que ya se venía utilizando hasta el momento. Ninguna de las otras opciones mostró una mejora significativa como para realizar un cambio en la estrategia seleccionada.

Probablemente estos resultados estén dados por el hecho de que en este problema tanto las heurísticas iniciales como las primales logran conseguir soluciones factibles de muy buena calidad, por lo menos para los tamaños de instancia que se están manejando; por lo que no vale la pena cambiar a una estrategia como la de *dfs* ya que funciona mejor para los casos donde no se tienen buena cotas primales.

Capítulo 7

Resultados

A modo de resumen se presenta el desempeño del algoritmo original, contra el desempeño del algoritmo luego de cada uno de los estudios que se fueron presentando a lo largo de este trabajo.

Los modelos a ser comparados son

- CPLEX-BC: La formulación de Dantzig, Fulkerson y Johnson adaptada al AngTSP con la desigualdad 3.36 y las variables y en la función objetivo. Se utiliza todo el framework CPLEX por defecto para el manejo del branch-and-cut y de los diferentes aspectos como las heurísticas y la búsqueda de planos de corte.
- AngTSP-BC: La formulación de Dantzig, Fulkerson y Johnson adaptada al AngTSP, implementada en CPLEX con las siguientes variaciones a las opciones por defecto del framework
  - *Farthest Insertion* con 3-opt como heurística inicial.
  - Las igualdades del sistema minimal incluidas en el modelo.
  - Las desigualdades válidas presentadas incluidas como planos de corte.
  - La extracción de las desigualdades de orden cúbico que se encontraban originalmente en la adaptación de la formulación al AngTSP.
  - La heuriśtica primal por defecto de CPLEX, ya que en el tamaño de instancia estudiado no resultó beneficioso otra elección.
  - La utilización de una estrategia de selección de branching basada en la priorización de las variables x por sobre las y.
  - La utilización de la estrategia *best bound* como regla para la selección de nodo.

El análisis se presenta sobre el tiempo de ejecución en segundos para resolver una instancia, fijando media hora como limite de ejecución. Cuando no se alcanza la optimalidad en el tiempo fijado, se reporta el gap alcanzado por el algoritmo en dicha instancia.

Se utilizaron veinte instancias generadas aleatoriamente con el mismo proceso que fueron generadas las que se utilizaron durante el transcurso del trabajo. De hecho, las seis primeras instancias son las que se utilizaron en cada una de las secciones. Cada una de las instancias fue resulta para las mismas combinaciones de parámetros que se manejó durante los anteriores análisis.

En cada una de las tablas correspondientes a cada instancia, se puede observar la cantidad de nodos a visitar para tener una idea de la magnitud del problema, y de como se comportan los algoritmos respecto del crecimiento de este parámetro.

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.08	8.81	399.10	(36.54 %)	(61.72 %)
AngTSP-BC	0.03	0.04	0.05	0.06	0.31

Cuadro 7.1: Tiempos para Instancia Random 1. 16 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	3.57	(29.34 %)	(61.40 %)	(83.43 %)	(100.00 %)
AngTSP-BC	0.71	97.66	201.32	79.65	128.66

Cuadro 7.2: Tiempos para Instancia Random 2. 35 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.33	(4.01 %)	(33.96 %)	(71.78 %)	(96.51 %)
AngTSP-BC	0.17	1.27	7.21	91.47	39.90

Cuadro 7.3: Tiempos para Instancia Random 3. 25 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.04	2.80	36.57	420.71	378.16
AngTSP-BC	0.02	0.02	0.02	0.03	0.15

Cuadro 7.4: Tiempos para Instancia Random 4. 12 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	2.32	(25.64 %)	(59.61 %)	(81.03 %)	(100.00 %)
AngTSP-BC	2.15	19.61	277.37	380.65	74.28

Cuadro 7.5: Tiempos para Instancia Random 5. 31 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	12.95	(36.13 %)	(65.93 %)	(86.78 %)	(100.00 %)
AngTSP-BC	11.35	(0.55 %)	(7.98 %)	(9.86 %)	(7.36 %)

Cuadro 7.6: Tiempos para Instancia Random 6. 46 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.17	224.84	(28.38 %)	(57.74 %)	(81.39 %)
AngTSP-BC	0.12	0.05	0.88	3.49	10.34

Cuadro 7.7: Tiempos para Instancia Random 7. 18 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	13.77	(39.52 %)	(65.45 %)	NA	(100.00 %)
AngTSP-BC	4.33	(4.56 %)	(10.18 %)	(9.74 %)	(16.46 %)

Cuadro 7.8: Tiempos para Instancia Random 8. 48 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.61	1081.00	(40.89 %)	(67.52 %)	(95.92 %)
AngTSP-BC	0.06	0.71	2.50	5.38	8.35

Cuadro 7.9: Tiempos para Instancia Random 9. 24 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	7.41	(33.71 %)	(65.81 %)	(90.73 %)	(100.00 %)
AngTSP-BC	4.69	268.10	(7.40 %)	(14.50 %)	(13.74 %)

Cuadro 7.10: Tiempos para Instancia Random 10. 47 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.01	4.98	115.14	1284.66	1209.47
AngTSP-BC	0.01	0.01	0.05	0.22	0.21

Cuadro 7.11: Tiempos para Instancia Random 11. 13 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.06	4.21	144.73	(10.79 %)	(31.91 %)
AngTSP-BC	0.02	0.02	0.04	0.06	0.15

Cuadro 7.12: Tiempos para Instancia Random 12. 14 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	14.04	(43.03 %)	(67.25 %)	(86.33 %)	(100.00 %)
AngTSP-BC	2.52	1159.41	(8.74 %)	(10.12 %)	(3.08 %)

Cuadro 7.13: Tiempos para Instancia Random 13. 48 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	3.01	(35.71 %)	(61.44 %)	(83.31 %)	(100.00 %)
AngTSP-BC	0.81	332.16	1080.07	1099.96	951.97

Cuadro 7.14: Tiempos para Instancia Random 14. 39 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.24	(9.07 %)	(44.92 %)	(74.42 %)	(97.97 %)
AngTSP-BC	0.18	6.73	21.03	36.54	68.46

Cuadro 7.15: Tiempos para Instancia Random 15. 26 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.60	56.17	(29.57 %)	(61.87 %)	(84.60 %)
AngTSP-BC	0.14	0.24	0.35	0.25	0.53

Cuadro 7.16: Tiempos para Instancia Random 16. 20 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	3.81	(30.54 %)	(57.69 %)	(82.19 %)	(100.00 %)
AngTSP-BC	2.26	159.97	1125.73	(2.48 %)	1015.01

Cuadro 7.17: Tiempos para Instancia Random 17. 36 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.01	0.72	2.43	10.66	19.72
AngTSP-BC	0.01	0.00	0.01	0.01	0.02

Cuadro 7.18: Tiempos para Instancia Random 18. 10 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	5.64	(39.67 %)	(65.42 %)	(85.45 %)	(100.00 %)
AngTSP-BC	0.90	183.48	(6.56 %)	(13.11 %)	(9.51 %)

Cuadro 7.19: Tiempos para Instancia Random 19. 40 nodos

Algoritmo \ alfa	1.0	0.75	0.5	0.25	0.0
CPLEX-BC	0.13	534.91	(27.26 %)	(54.29 %)	(72.04 %)
AngTSP-BC	0.07	0.40	1.03	2.03	2.38

Cuadro 7.20: Tiempos para Instancia Random 20. 17 nodos

Sin mayor análisis, debido a que ya se fueron justificando las elecciones de las diferentes opciones, se puede observar en todas las tablas las notorias mejoras realizadas al algoritmo. Cada uno de los diferentes aspectos contribuyó a convertir muchas instancias irresolubles(en media hora) en resolubles, algunas resolviéndose en tiempos de ejecución muy bajos. Sin duda, el aspecto que mayor injerencia tuvo en estas mejoras fue el estudio poliedral realizado sobre la formulación.

Capítulo 8

Conclusiones y trabajo futuro

Luego de finalizar con todo el trabajo realizado, hay varias observaciones importantes a recalcar, algunas de las cuales ya se fueron esbozando parcialmente en cada una de las secciones presentadas.

En primer lugar, una conclusión bastante lógica pero que no por eso se debe pasar por alto, es el alto impacto que tiene el estudio particular del problema.

Las herramientas de resolución general son muy buenas y el paquete CPLEX se mostró muy efectivo en todos los análisis realizados, de hecho se pudieron resolver instancias sin estudios particulares sobre el problema, que no se hubiesen podido resolver con mera fuerza bruta. Sin embargo, fue el estudio particular sobre el problema presentado lo que generó un cambio importante en la calidad del algoritmo, transformando en resolubles instancias que se habían presentado desafiantes al comenzar el estudio. Es importante destacar que de todas formas, todo el estudio particular sobre este problema, fue acompañado por las herramientas generales provistas por CPLEX que se traducen en un beneficio no menor en temas tan importantes como el manejo del árbol de branch-and-cut y la resolución de la relajación lineal de cada nodo. El desarrollo que tiene CPLEX hace que estas tareas se encuentren optimizadas y resulten de gran ayuda.

Con respecto a la formulación original utilizada, fue importante poder corroborar el comportamiento de todas las opciones de modelos presentes para saber en cual enfocarse. Resulta complicado realizar un formulación totalmente nueva a este problema, cuando el mismo es una variación de uno de los problemas más estudiados en investigación operativa, además de que parece bastante lógico apoyarse en los modelos que se desarrollaron, estudiaron y perfeccionaron durante más de 50 años. De todas maneras, fue importante ver el desempeño de todos los modelos estudiados adaptados a este nuevo problema, porque si bien los problemas guardan una clara relación, podía suceder que la relación de calidad entre los modelos para el Problema del Viajante de Comercio no sea extrapolable al Problema del Viajante de Comercio Angular.

En cuanto a las heurísticas, resultaron muy importantes para poder encontrar cotas primales de calidad. En muchos problemas la mayor dificultad pasa por encontrar una solución factible que sirva como cota primal para lograr podas interesantes del árbol de branch-and-cut. En este caso ese inconveniente no se presentó gracias a la performance de las heurísticas utilizadas, de hecho se pudo observar que en muchas de las instancias utilizadas las heurísticas llegaban o bien a la solución óptima, o bien a cotas primales muy cercanas al óptimo, haciendo que el problema principal del algoritmo pasase por el ajuste de las cotas duales para lograr el certificado de optimalidad.

En las heurísticas iniciales fue muy importante el trabajo realizado por las metaheurísticas de búsqueda local. Dichas búsquedas locales proporcionaron muy buenas soluciones con tiempos de ejecución muy bajos, porcentualmente casi nulos respecto de toda la ejecución del branch-and-cut.

Sobre las heurísticas primales se vuelve a mencionar el hecho de que las que se presentaron no tuvieron mejores resultados que las heurísticas primales que proporciona CPLEX por defecto, pero se pudo observar que esto se debe al tamaño de instancia con la que se pudo trabajar, en las cuales las heurísticas iniciales ya tenían un muy buen desempeño. Las pruebas que se realizaron

con instancias de mayor tamaño, y con instancias donde no se utilizaba las iniciales, mostraron que es importante la inclusión de heurísticas primales particulares para el problema.

El estudio poliedral que se realizó sobre el problema fue lo que marcó las notables mejoras respecto del algoritmo inicial.

Resultó sumamente importante lograr la caracterización de las igualdades del sistema minimal. Las mismas restrigen la dimensión de búsqueda de soluciones a la menor dimensión posible del espacio donde viven los puntos enteros que se quieren analizar. Esto resulta en la eliminación de muchos puntos fraccionarios que eran visitados innecesariamente cuando se resolvían las relajaciones lineales de cada nodo.

Además, la búsqueda de las igualdades del sistema minimal llevó también a encontrar desigualdades válidas útiles para el modelo.

Si bien no fue demostrado que todas las igualdades del modelo eran las presentadas, se logró una buena confianza en las mismas mostrando que sí eran las igualdades deseadas en instancias de pocos puntos. Además, más allá del aspecto teórico de caracterizar el sistema minimal, no cabe duda que las igualdades que se utilizaron resultaron importantisimas para mejorar la calidad del algoritmo.

Por el lado de las desigualdades válidas, también se pudo observar que su estudio e inclusión fue fructífero mejorando considerablemente los tiempos de ejecución.

Algo muy importante a destacar con respecto al estudio poliedral, es lo interesante que fue la interacción con el software PORTA para este punto. Dicha interacción proporcionó datos y formas de ver las cosas que no hubiesen sido posible de otra manera.

También dentro del estudio poliedral, es muy interesante destacar la mejora que se logró al sacar desigualdades en vez de incluirlas. Resultó importante volver sobre el modelo original para darse cuenta que las últimas mejoras agregadas, las igualdades encontradas, hacían obsoletas muchas de las desigualdades que el modelo original poseía para ser correcto. Esto también mostró lo importante que es para el algoritmo el tiempo que se tarda en resolver cada relajación lineal, ya que la redundancia de las desigualdades extraídas no tiene influencia en otro aspecto de la ejecución.

También se menciona lo importante que es analizar todos los aspectos posibles del algoritmo, ya que algo bastante simple de implementar y analizar como la estrategia de branching del árbol, logró también una mejora en los tiempos de ejecución.

En cuanto al trabajo futuro, es notable que al realizar un trabajo como este, la combinatoria producida por las diferentes decisiones que se tomaron en cada una de las secciones hace que sea imposible poder analizar cada una de las combinaciones de opciones posible. Esto hace que haya varios aspectos interesantes sobre los cuales trabajar en un futuro, a saber

Otros modelos Si bien el modelo utilizado fue elegido por su desempeño respecto de los demás, la elección de otro modelo inicial hubiese resultado en un algoritmo completamente diferente que, en su totalidad, podría haber resultado en uno mejor que el que se desarrolló durante este trabajo. Sería interesante ver cómo modelos que utilizan otros tipos de variables conducen a estudios poliedrales completamente diferentes al realizado.

Demostraciones teóricas En varios de los análisis presentados, se esbozan resultados que podrían ser demostrados rigurosamente. Dos de los más importantes son el resultado sobre la dimensión de la cápsula convexa de las soluciones factibles y las demostraciones de facecitud de las desigualdades válidas.

Estas demostraciones no se reducen a un mero aspecto teórico, sino que la realización de las mismas puede llevar a resultados prácticos útiles como, por ejemplo, el hallazgo de más igualdades o desigualdades válidas.

Desigualdades válidas y problemas de separación En todas las instancias que se resolvieron, se pudo observar que el problema se encontraba desde el lado dual. Es decir, no fue complicado encontrar la solución óptima, o configuraciones muy cercanas a ella, pero si resultó más difícil tener buenas cotas duales para lograr el certificado de optimalidad. Esto hace pensar que para un trabajo futuro lo más importante sería poner el esfuerzo en este punto, generando más familias de desigualdades válidas que fortalezcan el modelo para lograr relajaciones lineales lo más cercanas posible a la cápsula convexa.

Otro punto importante sobre las desigualdades válidas es en cuanto a los algoritmos de separación. La mayoría de los propuestos en este trabajo hacen uso de la fuerza bruta, salvo por pequeñas mejoras en algunas de las familias. Esto se encuentra de esta forma ya que para las instancias que se aspiraba a resolver no presentaba ningún problema. Sin embargo, sería importante pensar en cambios de estos algoritmos si se quiere apuntar a resolver instancias con bastantes más puntos en donde algoritmos de separación con complejidades polinomiales altas puedan empezar a afectar la performance del branch-and-cut.

Otros aspectos Existen otros aspectos variados que son interesantes para tener en la mira. Por ejemplo, uno de ellos es la optimización de la implementación. Algunas decisiones de implementación tienen un alto impacto en los tiempos de ejecución. Invertir tiempo en el desarrollo de mejores implementaciones puede ser una buena idea para lograr mejores tiempos de ejecución.

Otro punto que es también una cuestión de implementación, es el hecho de utilizar ciertas variables como continuas o como discretas. En este trabajo se puede observar que al incluir las igualdades del sistema minimal ya no hace falta que las variables de ángulo sean binarias, pudiendo ser continuas.

Bibliografía

[1] Achterberg, Koch, and Martin. Branching rules revisited. *Operations Research Letters*, (33):42–54, 2005.

[2] Alok Aggarwal, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *In Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 29:221–229, 1997.

[3] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[4] G Dantzig, R Fulkerson, and S Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

[5] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[6] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984.

[7] M. Grötschel, M. Jünger, and G. Reinelt. Calculating exact ground states of spin glasses: A polyhedral approach. pages 325–353. 1987.

[8] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[9] Martin Grötschel and Manfred Padberg. On the symmetric travelling salesman problem i: inequalities. *Math. Program*, (16):265–280, 1979.

[10] Martin Grötschel and Manfred Padberg. On the symmetric travelling salesman problem ii: lifting theorems and facets. *Math. Program*, (16):281–302, 1979.

[11] Martin Grötschel and Manfred Padberg. Ulysses 2000: In search of optimal solutions to hard combinatorial problems, 1993.

[12] Gregory Gutin, Abraham Punnen, Alexander Barvinok, Edward Kh. Gimadi, and Anatoliy I. Serdyukov. The traveling salesman problem and its variations, 2002.

[13] Steven M LaValle. Planning algorithms, 2004.

[14] Adam N. Letchford and Andrea Lodi. Polynomial-time separation of simple comb inequalities. pages 93–108. Springer, 2002.

[15] J. Lysgaard and Handelshøjskolen i Århus. Institut for Driftsøkonomi og Logistik. *CVRPSEP: A Package of Separation Routines for the Capacitated Vehicle Routing Problem.* Working paper // Department of Management Science and Logistics, Faculty of Business Administration, the Aarhus School of Business. Institut for Driftøkonomi og Logistik, Handelshøjskolen i Århus, 2003.

[16] Jens Lysgaard, Adam N. Letchford, and Richard W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:2004, 2003.

[17] André César Medeiros and Sebastián Urrutia. Discrete optimization methods to determine trajectories for dubins vehicles. *Electronic Notes in Discrete Mathematics*, 36:17–24, 2010.

[18] Temel Oncan, Kuban Altinel, and Gilbert Laporte. Invited review: A comparative analysis of several asymmetric traveling salesman problem formulations. *Comput. Oper. Res.*, 36(3):637–654, March 2009.

[19] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, February 1991.

[20] Ketan Savla, Francesco Bullo, and Emilio Frazzoli. On traveling salesperson problems for dubins' vehicle: stochastic and dynamic environments, 2005.

[21] L. A. Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998.