UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

FACULTAD DE INGENIERIA

Modelización y Resolución Exacta vía Programación Lineal Entera del "Prize-Collecting Steiner Tree Problem"

Investigación Operativa

Martín Berguer Centurión

Programa de grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Abril de 2017

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

FACULTAD DE INGENIERIA

Modelización y Resolución Exacta vía Programación Lineal Entera del "Prize-Collecting Steiner Tree Problem"

Investigación Operativa

Martín Berguer Centurión

Tesis de Ingeniería presentada al Programa de grado en Ingeniería en Computación, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Ingeniero en Computación.

Directores:
Dr. Ing. Franco Robledo Amoza
Dr. Ing. Pablo Romero Rodríguez

Agradecimientos

Los resultados de este proyecto están dedicados a todas aquellas personas que de alguna forma fueron parte de su culminación. A mis tutores Franco Robledo y Pablo Romero por la confianza plena durante todo el desarrollo del proyecto. A Pablo Olivera y Cristian Bauza por las horas que compartimos en este proyecto. A Paula por todo el apoyo brindado desde el comienzo. Y en especial a Iván, Nancy, Alexander y Mabel, nada podría haber sido posible sin ellos.

Man is born as a freak of nature,
being within nature and yet
transcending it. He has to find
principles of action and
decision-making which replace
the principles of instincts.

Erich Fromm

RESUMEN

En este documento se estudia el problema del *Prize-Collecting Steiner Tree (PCST)*, el cual pertenece a la clase de problemas NP-Difíciles. Para comenzar, se presenta un estudio de las técnicas de resolución exacta existentes en la actualidad, seleccionando cinco formulaciones del problema utilizando técnicas de Programación Lineal Entera.

En esta investigación se introduce una nueva formulación para resolver el PCST de forma óptima. Se realizaron dos implementaciones, el nuevo modelo propuesto en esta investigación y una de las cinco formulaciones mencionadas previamente. Además, se realiza un estudio comparativo entre ambas implementaciones así como otras implementaciones basadas en algoritmos de aproximación, también capaces de resolver el PCST. Por otro lado, se realiza un estudio práctico referente a la relajación de ambos modelos, en busca de cotas inferiores de la solución óptima.

Por último, se estudia la aplicación del modelo en una numerosa variedad de redes que representan situaciones reales, las cuales han sido estudiadas en una cantidad considerable de investigaciones referentes al tema y se realiza una aplicación práctica en la red de generación de energía eléctrica por parte de los parques eólicos dispuestos en el territorio uruguayo.

Los resultados del análisis experimental realizado son bien alentadores, muestran que ambos modelos se comportan correctamente, cumpliendo con el objetivo de llegar a la solución óptima.

Palabras claves:
Problema de Steiner, PCST, ILP, NP-Difícil, CPLEX.

Tabla de contenidos

Lista de siglas

API	Aplication Programming Interface
DIMACS	Discrete Mathematics and Theoretical Computer Science
DOE	Despacho Óptimo de Energía
EP	Entorno de Pruebas
FING	Facultad de ingeniería
GAP	Distancia o diferencia excesiva que existe entre dos valores
GSEC	Generalized Subtour Elimination Constraints
ILP	Integer Linear Programming
MCF	Multy-Commodity Flow
MWCSP	Maximum-Weight Connected Subgraph Problem
NDRC	Non-Delayed Relax-and-Cut
NWST	Node Weighted Steiner Tree Problem
PCM	Prize Collecting Model
PCNWST	Prize-Collecting Node-Weighted Steiner Tree
PCNWSTP	Tree Problem
PCST	Prize Collecting Steiner Tree
PL	Prorgramación Lineal
PLE	Programación Lineal Entera
Prizewer	Prize Collecting Model Viewer
RO	Robust Optimization
SPWST	Single Point Weighted Steiner Tree Problem
SCF	Single Commodity Flow
STP	Steiner Tree Problem
Udelar	Universidad de la República

Capítulo 1

Introducción

1.1. Motivación

La planificación energética es un tema estratégico de los operadores de energía eléctrica de cada país.

En Uruguay el advenimiento de energías renovables y otras fuentes de generación adicionales a las Represas Del Palmar, Rincón del Bonete, Baygorria, y Salto Grande (compartida con la República Argentina) ha generado que la problemática tome altísima relevancia, buscando integrar en la planificación del despacho de carga esta nueva realidad y sus perspectivas futuras de crecimiento. En Uruguay las herramientas de Planificación Energética básicamente se descomponen en tres, según el horizonte: corto plazo (menos de 24 horas), mediano plazo (entre 48 horas y dos semanas), y largo plazo (3 meses aproximadamente, o más).

En la Facultad de Ingeniería, un grupo de Investigadores del Dpto. de Investigación Operativa y del IMERL está participando en un proyecto financiado por la Agencia Nacional de Investigación e Innovación denominado: "Planificación estocástica óptima para la generación y acumulación diaria de energía, integrada a políticas de control en Smart Grids".

Bajo este contexto dicho proyecto busca los siguientes objetivos:

I Desarrollar e implementar un modelo unificado de optimización estocástica y algoritmos para la resolución del problema de *Despacho Óptimo de Energía (DOE)* a corto plazo (24 horas), para sistemas con porcentajes significativos de energías renovables no acumulables, que integra elementos para acumulación y la posibilidad de afectar dinámicamente el

consumo mediante cambios en los precios.

II Generar escenarios prospectivos representativos de las distintas inversiones y/o políticas a consideración de las autoridades competentes, como: bombear agua hacia embalses de centrales hidráulicas, nuevas centrales de bombeo, uso residencial de baterías y/o dispositivos telecontrolados para ciertos tipos de electrodomésticos.

III Resolver las instancias asociadas a esos escenarios para evaluar la conveniencia económica para el ecosistema eléctrico nacional (social welfare) de las inversiones y políticas bajo consideración. Recientemente el país ha incrementado en forma sostenida el porcentaje de potencia instalada proveniente de energías renovables. Se espera que a fines de 2017 el 90 % de la energía anualmente consumida provenga de fuentes renovables, entre ellas: hidráulica, eólica, solar y biomasa. La imposibilidad práctica de acumular energía eólica y solar, debe compensarse con la utilización racional de otras formas de energía, pero normalmente esto no es suficiente ni económico. El notable avance nacional en las Tecnologías de la Información y la Comunicación abre espacio para explorar alternativas más eficientes, como impulsar un consumo eléctrico inteligente mediante la fijación dinámica de precios, o incluso recurrir a la acumulación residencial como fuente de potencia hacia la red pública. El problema es intrínsecamente estocástico.

Dentro de los productos a obtener en este proyecto se espera:

(a) Motor de optimización estocástica que integre generación y demanda en un mismo DOE.

(b) Instancias del modelo correspondientes a los escenarios prospectivos de políticas bajo evaluación.

(c) Resultados cuantitativos para el retorno esperado de esos escenarios.

El proyecto es altamente complejo por las múltiples variables en juego y subproblemas que surgen de su análisis. El equipo de FING está en plena ejecución del proyecto con perspectivas de finalización a mediados de 2018. Ahora bien, como una de las partes vinculadas al proyecto, surge el problema de localización óptima de un conjunto de generadores (eólicos) dentro de un

parque preestablecido (un campo). Es así que el modelo conocido como "Prize-Collecting Steiner Problem (PCST)" se considera adecuado para modelar este problema (en una variante simplificada) de asignación de generadores a lugares factibles e interconectarlos topológicamente a costo mínimo, con penalizaciones por el no uso de aquellos sitios que quedan "vacíos" (no se colocarán generadores). Es así que este proyecto de grado ataca técnicas de solución basadas en algoritmos exactos, para resolver el PCST. Como un aporte más desde el enfoque académico a técnicas de optimización que resuelvan una componente de un problema mayor como lo es la Planificación Óptima del Despacho de carga teniendo en cuenta políticas prospectivas. Este proyecto de grado busca aportar soluciones al problema de diseñar topologías de despliegue de parques eólicos, pensando más que nada en la expansión de este tipo de fuentes de generación en el Uruguay, y la apuesta que desde el Gobierno se ha hecho con políticas de largo plazo al respecto. Sus resultados serán de ayuda comparativa para las soluciones que diseñen los integrantes del proyecto mayor, ejecutado por los investigadores de FING.

El PCST también aplica para otros rubros de uso general tales como el gas, y telecomunicaciones. En los últimos años, el modelo clásico de negocio se ha alterado de forma notoria debido a inversionistas en busca de nuevos horizontes a nivel mundial, sumado a la creciente demanda de la sociedad moderna, altamente comunicada de alguno de estos bienes mencionados. De este modo, poder planificar de forma eficiente un escenario con varios potenciales clientes, cuya demanda es conocida, y poder estudiar caso a caso la posibilidad de conectarlos de alguna forma con un enlace previendo su costo, puede ser muy provechoso, sin tener que romper las calles de la ciudad, ahorrando en dinero y tiempo. Las compañías de alguno de los rubros mencionados tiene como objetivo obtener algún tipo de ganancia, y en ese proceso existen dos grandes actores: Por un lado se encuentran los clientes, que son los que le darán ganancias (en términos monetarios) a la empresa. Por otro lado, la red que intercomunique estos clientes debe ser diseñada de forma tal que sea eficiente en términos de costo. Es ese intercambio, que introduce un problema de compromiso del punto de vista de ingeniería, entre maximizar la suma de las ganancias que se obtienen de los clientes que son conectados, en contraste a la minimización del costo que es necesario para conectarlos.

1.2. El problema Prize-Collecting Steiner Tree

De manera abstracta, los problemas del tipo "Prize-Collecting" involucran situaciones donde hay varios puntos con un valor de demanda, cuyo propósito es pertenecer a una estructura, y el objetivo del problema es encontrar la estructura de costo mínimo. La clave está en que algunos de los puntos tienen un costo de alcance muy elevado como para ser incluidos, y se toma la decisión de no utilizarlos siempre que se pague una penalización. En este punto, se tratará el estudio del "Prize-Collecting Steiner Tree", en el cual la estructura mencionada anteriormente puede modelarse como un grafo $G = (V, E)$ no dirigido, con un costo no negativo c_e asociado a la arista e, que simboliza el costo de elegir esa arista. Y además un costo no negativo π_v (llamadas prize, o ganancia en español) en cada uno de los nodos, representando la penalización del nodo v. Dado cualquier subconjunto (E') de aristas, se denota como $V(E')$ al conjunto de nodos que actúan como extremo en alguna de las aristas de (E') y diremos que (E') recubre a $V(E')$. Dado cualquier subconjunto (V') de nodos, se denota $\overline{V'}$ como su complemento $(V - V')$.
En PCST se debe seleccionar un conjunto de aristas T tal que $(V(T), T)$ sea un árbol y así minimizar el costo combinado:

$$costo = \sum_{e \in T} c_e + \sum_{v \in \overline{V(T)}} \pi_e = c(T) + \pi(\overline{V(T)})$$

Donde la notación $C(T)$ y $\pi(\overline{V(T)})$ siguen la convención habitual, cuando T es un vector.

El objetivo es minimizar el costo total de las aristas del árbol, así como la penalización de los nodos a los que no se alcanzó a recubrir Archer et al. (2011).

1.3. Un poco de Historia

El Prize-Collecting Steiner Tree Problem es una generalización del *Steiner Tree Problem (STP)*, la historia de este último presenta diversas particularidades, en algunas ocasiones ha sido completamente olvidado y luego redescubierto.

En el año 1643, Pierre de Fermat publicó su trabajo titulado "Method for Determining Maxima and Minima and Tangents to Curved Lines", en lenguaje

latín, y allí se planteaba lo siguiente: "datis tribus punctis, quartum reperire, a quo si ducantur tres rectae ad data puncta, summa trium harum rectarum sit minima quantitatis", una traducción al texto puede ser "dados tres puntos, encontrar un cuarto que, si se dibujan tres líneas que interconecten esos puntos, la suma de largo de dichas líneas sea de costo mínimo". Este problema se conoce como "el triángulo de Fermat" (Figura 1.1).

Figura 1.1: Triángulo de Fermat

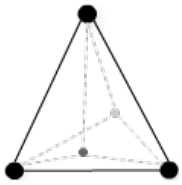
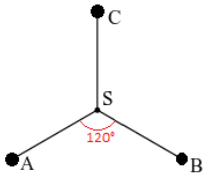


Figura 1.2: Solución



La primera solución que conocemos del problema la planteó Evangelista Torricelli. Su planteo es por construcción y su fundamento es puramente geométrico. Posteriormente se comprobó que en este caso la solución (o el punto) que buscamos es aquel que forma exactamente 120 grados con todos los vértices Figura 1.2. Al generalizar el problema para cualquier grafo de tamaño n, adicionalmente a lo anterior, se respeta que todos los puntos o vértices del grafo solución tendrán grado tres.

No es claro cómo el paso del tiempo hizo que se lo llamara "Problema de Steiner", sin embargo luego de sucesivas soluciones, desapariciones, y reapariciones del problema, Courant and Robbins (1941) publican el libro llamado "What is Mathematics?" con una sección dedicada a explicar el problema de Fermat-Torricelli bajo el subtítulo "Steiner Tree Problem"[1][2], y desde ese momento, las publicaciones que hacen referencias a este problema, lo mencionan como "Steiner Tree problem", citando esta sección del libro Brazil et al. (2014).

[1]Los autores lograron obtener uno de los manuscritos privados de Steiner, publicaron los resultados allí mencionados, generando la confusión.

[2]Posteriormente pudo comprobarse la veracidad de lo anterior ya que Krarup and Vajada (1997) observaron un error en la página 358 del libro, el mismo error que cometió Steiner en sus manuscritos.

1.4. Modelos de Optimización

Citando nuevamente la frase presentada al comienzo de esta tesis, Erich Fromm resume un aspecto fundamental de la consciencia humana, las acciones que tomamos deben tener en cuenta el contexto de todo lo que nos rodea y principalmente el ser humano debe tomar decisiones para poder sobrevivir. En el libro "Metaheuristics from design to implementation" Talbi (2009), se plantea la toma de decisiones como un proceso de cuatro estados: {formular, modelar, optimizar, implementar}. Pero no lo plantea como etapas ordenadas, sino iterativo, de modo que siempre se pueda mejorar el modelo de optimización hasta encontrar una solución aceptable al problema.

Es posible distinguir varios grupos de modelos de optimización. Esta investigación se centra en los modelos de programación matemática, más concretamente en la resolución del PCST modelado como un *Problema de Programación Lineal Entera (PPLE, o IPL por sus siglas en inglés).*
Un modelo de *Programación Lineal (PL)* respeta el siguiente formato:

$$PL = \begin{cases} \text{Min / Max :} & \\ & c * x \\ \text{Sujeto a:} & \\ & A * x \geqslant b \\ & x \geqslant 0 \end{cases}$$

En programación lineal tanto la función objetivo que se busca optimizar, como las restricciones son funciones lineales. La programación lineal es uno de los planteos más satisfactorios para resolver problemas de optimización.
De hecho para problemas de optimización lineal continua existen algoritmos exactos eficientes, como el *método simplex* y el *método de punto interior*.

Método Simplex

El método simplex se introduce en Dantzig (1947) como una herramienta para resolver problemas de programación lineal, es sumamente eficiente, básicamente consiste en partir de una solución básica factible y recorrer todos los vértices de la región factible. Con el pasar de los años, se le han realizado algunas variaciones como la introducción de la forma canónica, donde todas las restricciones deben presentar una desigualdad y además las variables deben ser positivas, aunque sigue respetando las directivas originales. Esto quiere decir

que cualquier problema de PL se puede formular en la forma canónica, como en la original indistintamente.

Este método fue un pilar muy importante para grandes aplicaciones matemáticas que surgieron posteriormente. Existen varios algoritmos y herramientas que se pueden derivar a partir de la programación lineal; por ejemplo, la teoría de flujo máximo y corte mínimo formulada por T. E. Harris y quienes plantearon el primer algoritmo que se conoce fueron L. R. Ford, Jr. y D. R. Fulkerson Schrijver (2002).
Por otro lado la Teoría de Juegos formalizada por Von Neuman en la década del 30, en 1944 el propio Von Neuman con Oskar Morgenstern publican "Theory of Games and Economic Behavior" Neumann and Morgenstern (1944). La teoría tuvo un crecimiento sustancial debido a la guerra fría y su aplicación en el ámbito militar. En 1950, John Forbes Nash defiende su tesis de doctorado Nash (1950), estableciendo lo que hoy se conoce como "Equilibrio de Nash".

Una aplicación de la teoría de juegos es el método MiniMax. En 1996, este algoritmo fue el que se implementó para el duelo entre la computadora de ajedrez Deep Blue y Garry Kasparov que no solo marcó un hito desde el punto de vista tecnológico matemático, sino que mostró los mecanismos de inteligencia que las computadoras son capaces de manejar - Deep Blue.

1.5. Problemas NP-Difíciles

En términos de complejidad computacional, se dice que un algoritmo es eficiente si, para cualquier instancia de un determinado problema, se puede encontrar una solución en tiempo polinomial.

En contraste a estos, existen algoritmos de complejidad sobrepolinomial, que se comportan de forma distinta; a medida que aumenta el tamaño del problema, se vuelve muy difícil de solucionar, independientemente de la capacidad de cómputo, esto provocó el afán de los investigadores por encontrar soluciones de tiempo polinomial a diversos problemas, aunque han habido diversas dificultades como se verá en este capítulo.

Llamaremos *problemas de decisión*, a aquellos que tienen como objetivo retornar un resultado de carácter booleano (sí o no), y en este contexto se define lo siguiente:

***Definición* 1.1** *Digamos que un problema es C-Difícil si es al menos tan*

difícil como cualquier problema de C, y es C-Completo si además pertenece a C.

***Definición* 1.2** *Llamemos P a la clase de problemas que admiten solución en tiempo polinomial con la entrada, por una máquina de Turing determinista.*

***Definición* 1.3** *Llamemos NP a la clase de problemas de decisión que admiten una verificación de una instancia positiva en tiempo polinomial.*

En otras palabras, *NP* es la familia de problemas de decisión que admite un certificado positivo con una máquina de Turing no determinista.

***Definición* 1.4** *Un problema A se reduce a B si existe una función inyectiva $\pi : A \rightarrow B$ que traduce toda instancia de A en alguna instancia de B en una cantidad polinomial de operaciones. Si A se reduce a B, entonces B es al menos tan difícil como A. Técnica para probar que X es NP-Completo: Probamos que $X \in NP$. Elegir $A \in$ **NP-Difícil** y hallar $\pi : A \rightarrow X$.*

De estas definiciones se puede inferir que *P* está incluido en *NP*, sin embargo al momento no se ha podido demostrar que $P \neq NP$. Se podría especular sobre ello, si se demostrara que existe un algoritmo de tiempo polinomial para resolver un problema *NP-Completo*, y en tal caso todos los problemas en *NP-Completo* podrían ser resueltos en tiempo polinomial a través de las transformaciones.

Cook (1971) demostró que el problema MAX-SAT es *NP-Completo*.

En MAX-SAT, interesa saber si dada una expresión booleana (con variables y sin cuantificadores), sus variables tienen asociada una asignación de valores que hace que la expresión general sea verdadera. La demostración de este problema es de suma importancia en la teoría de complejidad; significa que no hay ningún sistema de aproximación de tiempo polinomial a este problema a menos que *P=NP*.

También es necesario tener presente que un problema del tipo *NP-Difícil* es tan complejo como cualquiera de la clase *NP*. Un año después Karp (1972) utilizó la reducibilidad de Cook para dar la lista de los primeros 21 problemas *NP-Completo*, entre los que se encontraba el problema de árboles de Steiner (STP por sus siglas en inglés); y dado que este es un caso particular del PCST, es posible afirmar que el PCST también lo es.

Se recomienda la lectura de Chapovska and Punnen (2006) donde se presentan algunas variantes del PCST, y además se prueba que todas sus variantes también pertenecen a la clase NP.

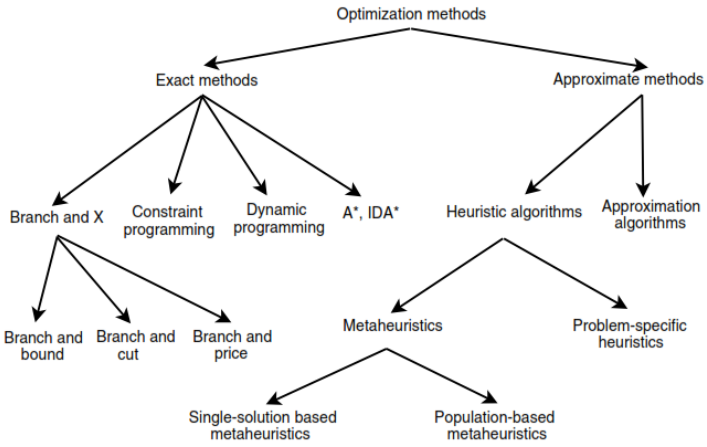
1.5.1. Estrategias

Todos los problemas *NP-Difíciles* pueden ser resueltos utilizando la "fuerza bruta". Sin embargo conforme crece el tamaño del problema, crece el tiempo de ejecución alcanzando cifras desmesuradas, en algunos casos con problemas aún pequeños.

Esta problemática dio lugar al surgimiento de metodologías, que sirven de guía a la hora de trabajar con problemas *NP-Difícil*. En la Figura 1.3 se aprecia un esquema de los distintos métodos existentes de resolución de problemas, identificando dos grandes conjuntos; los *métodos exactos* y los *métodos aproximados*.

Como se verá a continuación, el mecanismo de resolución que utilizan es muy diferente, brindando un abanico muy grande de posibilidades. Donde el método a utilizar dependerá del problema a resolver.

Figura 1.3: Métodos clásicos de optimización Talbi (2009)



Algoritmos heurísticos

Dentro del grupo de algoritmos aproximados, se encuentran las *Heurísticas*; como su nombre[3] lo indica, tienen como objetivo "descubrir" la solución. Son

[3] ***heurístico, ca***: Del gr. *εύρίσκειυ* heurískein 'hallar', 'inventar'. Real Academia Española, 2017.

un método de solución a problemas *NP-Difícil*, y constituyen una opción muy interesante permitiendo alcanzar resultados muy cercanos al óptimo en forma eficiente en muchos casos, aunque sin garantizar la optimalidad.
Las heurísticas se clasifican de acuerdo al método utilizado:

1. **Constructivo:** Partiendo de una solución vacía, van agregando componentes hasta obtener una solución completa. Generalmente son de rápida ejecución, pero no dan buenos resultados.

2. **Búsqueda local:** Partiendo desde una solución, se comienza un proceso iterativo que busca llegar a una mejor solución en cada caso.

El tema adquiere más interés al considerar las *Metaheurísticas*. Estos son algoritmos que combinan heurísticas básicas en esquemas de alto nivel para realizar eficientemente y efectivamente la exploración de un espacio de búsqueda.
Las Metaheurísticas son consideradas como heurísticas con un grado de abstracción superior, pueden resolver una variada cantidad de problemas "adaptándose" en cierta forma a ellos.
Son versátiles, pueden utilizar simples procedimientos de búsqueda local, como procesos de aprendizaje complejos, así como también tienen la capacidad de hacer uso de otras heurísticas. Además de utilizarse como patrón general para la resolución de problemas, pueden adaptarse para hacer uso del conocimiento específico del dominio, así como utilizar la experiencia de búsqueda que ya tienen (almacenada en memoria) para mejorar el proceso de búsqueda de solución.

Su objetivo es explorar de manera eficiente el espacio de búsqueda con el fin de encontrar soluciones cercanas a la óptima. Esto último se relaciona estrechamente con otra característica de las metaheurísticas; es de fundamental importancia no quedar "atrapado" en una zona del espacio de soluciones. Salamon et al. (2002)

Existen variados criterios de clasificación, y esto probablemente sea debido a que en las últimas décadas han habido enormes avances para resolver problemas de la clase *NP-Difícil*.

Se enunciarán algunos de los criterios de los más afianzados y que a su vez han demostrado ser aplicables en diversos escenarios, dato que ha reivindicado a las metaheurísticas como una sólida alternativa a la hora de tratar con problemas *NP-Difícil* Pedemonte (2009).

1. **Métodos basados en población o en trayectoria**: Los primeros se caracterizan por el uso de un único conjunto de soluciones en cada paso, al que se le conoce como población. Mientras que en los métodos de trayectoria se utiliza una única solución, y el mecanismo de exploración se caracteriza por el uso de una única trayectoria.

2. **Inspiradas (o no) en la naturaleza (sistemas biológicos, físicos o sociales)**: Este es uno de esos casos donde el estudio y la observación de eventos cotidianos ayuda al desarrollo de una rama muy importante del mundo científico y más concretamente el área de optimización como es en este caso. El criterio es muy sencillo, en el caso de los inspirados en la naturaleza, la idea es abstraer al máximo la esencia y el comportamiento de los actores y plasmar cómo éstos interactúan con su ambiente y las condiciones o normas por las que se rigen. Mientras que los métodos que no están inspirados en la naturaleza, son más del estilo clásico, como puede ser un algoritmo de búsqueda.

3. **Con memoria o sin memoria**: Un mecanismo puede hacer uso o no de memoria. En caso de que use puede hacerlo a corto o largo plazo; la diferencia entre estas dos es identificar soluciones recientemente visitadas, o utilizar toda la experiencia almacenada desde el comienzo la búsqueda.

Estos son algunos de los criterios para clasificar metaheurísticas. Existen variadas aplicaciones de estas a problemas puramente matemáticos, como aplicaciones a situaciones reales, como se plantea en Pedemonte (2009) Talbi (2009).

Algoritmos de aproximación

Los algoritmos de aproximación son la otra opción que se dispone en la familia de algoritmos cuyo objetivo es encontrar una solución aproximada. Un ρ-algoritmo de aproximación, para un problema de optimización, es un algoritmo de tiempo polinomial que para todas las instancias del problema, produce una solución cuyo valor está a lo sumo dentro de un factor ρ de la solución óptima. Diremos que el ρ (también llamado razón o factor de aproximación) es el desempeño que podemos garantizar de dicho algoritmo Vazirani (2001).

Formalmente:

***Definición* 1.5** *Dada una instancia I de tamaño n, definimos:*

- *C(I) como el costo de la solución producida por el algoritmo de aproximación.*
- *C*(I) como el costo de la solución óptima.*

La clave se encuentra en la relación entre C(I) y C*(I), donde dependerá de la función objetivo del problema:

- C(I)/C*(I) debe ser pequeño en caso que sea **minimización**
- C*(I)/C(I) debe ser pequeño en caso que sea **maximización**

Definición* 1.6** *Se define el* ***cardinalidad *de un conjunto es el número de elementos que posee ese conjunto.*

Dado un conjunto A, el cardinal se simboliza mediante $|A|$.

***Definición* 1.7** *Diremos que un algoritmo de aproximación alcanza un factor $\rho(n)$ si para toda instancia I, $|I| = n$ si:*

$$max\left(\frac{C(I)}{C^*(I)}, \frac{C^*(I)}{C(I)}\right) \leqslant \rho(n)$$

Aunque se debe tener presente que existen algunos problemas *NP-Difíciles*, que no cuentan con las condiciones para encontrar dicho algoritmo Vazirani (2001) Williamson and Shmoys (2011).

En paralelo a esta investigación, Bauza and Olivera (2017) realizaron un estudio sobre el PCST pero aplicando algoritmos de aproximación para resolverlo. Su investigación es muy interesante la cual se tratará más en detalle en la Sección 2.3 y algunos temas referentes a su implementación en el Capítulo 5

Solución óptima

Algunas definiciones previas antes de abordar la problemática Talbi (2009)

Definición* 1.8** ***Big-O notation*: Un algoritmo tiene complejidad f(n) = O(g(n)), si existen constantes positivas* n_0 *y c tal que* $\forall n \geqslant n_0, f(n) \leqslant c * g(n)$

Definición* 1.9** *Un algoritmo es de* ***tiempo polinomial *si su complejidad es O(p(n)), con p(n) una función polinomial de grado n.*

Definición* 1.10** *Un algoritmo es de* ***tiempo exponencial (o sobrepolinomial) *si su complejidad es* $O(c^n)$*, con c una constante real mayor estricto que 1.*

Muchos de los problemas *NP-Difícil*, surgen en situaciones donde la realidad se ha encargado de plantear determinadas condiciones, y la decisión que se tome para ese determinado problema suele ser generalmente la que deje mejor partido. Más aún en casos donde existe un agravante económico y la decisión que se tome conlleva una carga monetaria elevada.

En situaciones de este estilo, conviene aplicar un método que dé como resultado la "mejor" solución, y con mejor, por supuesto no siempre significa la más económica, sino la que otorgue mejor relación costo beneficio.

El método llamado "fuerza bruta" tiene orden $O(n!)$ dado que prueba absolutamente todas las combinaciones posibles. En particular para problemas *NP-Difícil* produce lo que muchos autores denominan *"tiempo prohibitivo"*. Para prevenir que esto suceda, es necesario utilizar algún mecanismo que en base a determinadas condiciones sea capaz de reducir el espacio y no probar con todos los casos, así es como surgen los algoritmos exactos Talbi (2009).

Los avances más significativos en el área, datan de mediados del siglo XX, y eso comprende por igual a todos los métodos que presentaremos a continuación:

Programación dinámica: Es un algoritmo recursivo que divide el problema original en problemas más simples. Respeta el esquema *Bottom Up*; resuelve

primero los problemas más pequeños y a medida que continúa la ejecución del algoritmo, se utilizan estos resultados para resolver el problema más grande. El pilar fundamental de la programación dinámica lo estableció Richard Bellman en la década de 1950, cuando enunció el principio de optimalidad; "dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez óptima". Este principio se aplica mediante la poda de algunas secuencias parciales de decisión, que se sabe de antemano que no llevarán a la solución óptima.

Métodos Branch-and-*: Los algoritmos *Branch-and-Bound, Branch-and-Cut, Branch-and-Backtrack* y *Branch-and-Prize* son los predilectos a la hora de solucionar problemas de programación lineal entera. A grandes rasgos, todos operan de una forma similar, iteran sobre el espacio de soluciones hasta cumplir (o no) con cierta condición, lo que va acotando el criterio hasta llegar a la solución óptima.

1.6. Conclusiones

El objetivo de estudio es el PCST, dado que pertenece a la clase de problemas *NP-Completo*, en marco de la problemática planteada al comienzo fue necesario seleccionar alguna de las técnicas existentes para la resolución de problemas modelados en forma de Programación Lineal Entera.

De las diversas técnicas existentes, el pensamiento inicial consistía en atacar el problema con una solución a medida utilizando la técnica de Branch-and-Cut. Sin embargo luego de evaluar todas las posibilidades, se optó por utilizar le herramienta *CPLEX*[4], que provee una gran variedad de estrategias para resolución de problemas de programación matemática, entre ellas la técnica Branch-and-Cut. La decisión de utilizar esta herramienta al comienzo no era del todo convincente, pero investigando se notaron dos aspectos fundamentales:

- **Tiempo:** Al utilizar un framework que ya provee algunas funcionalidades necesarias, permite reducir el tiempo dedicado a codificar, y a su vez permite dedicarle mayor atención al objetivo de este proyecto, que es el de generar un modelo funcional que llegue al óptimo.

- **Errores:** De igual forma, en todo proceso de desarrollo existe el riesgo de introducir errores. Es cierto que esto también puede ocurrir al utilizar

[4]CPLEX es un software capaz de solucionar problemas de Programación Lineal, Programación Entera Mixta y Programación Cuadrática

CPLEX, pero con una probabilidad menor, ya que es un software que está desde hace varios años en el mercado, ha sido testeado, y es utilizado por grandes empresas en el mundo así como una gran parte de la comunidad académico-matemática.

- **Recursos:** CPLEX cuenta con mecanismos de procesamiento sincronizado, que permiten aprovechar al máximo los recursos disponibles del sistema como la memoria, y los hilos de procesamiento.

Por último, comentar que es muy performante: Hans Mittelmann realiza estudios sobre los solvers disponibles en el mercado, y en uno de los más recientes[5] menciona que CPLEX está entre las primeras posiciones. Además en la página oficial de IBM, se menciona la posibilidad de adquirir una licencia de estudiante.

Estos detalles, esclarecieron las incertidumbres iniciales y terminaron inclinando la balanza en favor de utilizar CPLEX.

1.7. Organización del documento

El documento continúa con un relevamiento del estado del arte de las técnicas aplicadas para la resolución del PCST, posteriormente detallan algunas de las formulaciones exactas existentes y una técnica de algoritmos aproximados en el Capítulo 2. En el Capítulo 3 se introduce un nuevo modelo exacto, propuesto en esta investigación con la finalidad de resolver el PCST. Además se presenta una segunda implementación, proveniente de uno de los modelos presentados en el Capítulo 2.

En el Capítulo 4, se detallan algunos aspectos de la implementación de ambos modelos. Luego, en el Capítulo 5 preparando lo que será el análisis de ejecución, se analiza la complejidad matemática de ambos modelos descriptos en el Capítulo 4 y además se detallan otras implementaciones con las que se compararán con ambos modelos. En el Capítulo 6 se estudia la comparativa de los resultados obtenidos al correr diversos casos de prueba y finalmente en el Capítulo 7 están presentes las conclusiones finales así como el trabajo a futuro.

[5] http://plato.asu.edu/ftp/lpsimp.html

Capítulo 2

Estado del Arte

Este capítulo incluye la revisión de la literatura, así como los diferentes enfoques de los métodos existentes similares a los aplicados en esta investigación.

2.1. Trabajos relacionados

En 1987 se introdujo el problema de árboles de Steiner con costos en los vértices, como un adicional a el ya conocido con costo en las aristas, el autor del artículo Segev (1987) nombró este problema *Node Weighted Steiner Tree Problem (NWST)*. El objetivo es minimizar el costo de los vértices sumados a la ponderación de las aristas. Segev además plantea un problema simplificado del anterior, llamado *Single Point Weighted Steiner Tree Problem (SPWST)*, donde hay un vértice especial que debe ser seleccionado y a su vez existen nodos especiales con costos de valor no positivo, y aristas de costo no negativo, que reflejan el haber seleccionado esos nodos. El detalle de tener un valor negativo en el costo de los nodos es un punto a favor, ya que al estar restando en la función objetivo, dan como resultado valores positivos. El SPWST es equivalente al R-PCST que mencionaremos más adelante Ljubić et al. (2006).

El PCST fue introducido por Balas (1989), sin embargo fue Bienstock et al. (1993) quien planteó el primer algoritmo de aproximación para el problema, logrando un factor 3, como un problema de programación lineal relajado. Luego Goemans and Williamson (1995) lograron obtener un algoritmo con un factor $2 - \frac{1}{n-1}$, que tiene un orden $O(n^3 log(n))$, siendo n el número de vértices del grafo.

En el año 2000, Johnson et al. plantearon una solución al problema con factor

exacto 2, pero con un orden de ejecución menor $O(n^2 log(n))$.

Finalmente aparece Feofiloff et al. (2007)[1] con el mismo tiempo de ejecución pero con un factor $2 - \frac{2}{n}$ Archer et al. (2011).

Canuto et al. (2001) desarrolló una Metaheurística de búsqueda local que hace uso de múltiples inicios con perturbación. Para la solución factible inicial se construye invocando una variante aleatoria del algoritmo de Goemans and Williamson (1995) que modifica la penalidad de los nodos y en la iteración utiliza el método de exploración local GRASP[2]. Hasta aquí se ha comentado principalmente algoritmos de aproximación y heurísticos, pero en Lucena and Resende (2004) se presenta un método para obtener cotas inferiores, el modelo se basa en una formulación binaria que tiene un número exponencial de restricciones de eliminación de subcaminos[3]. Su implementación utiliza un enfoque de planos de corte.

Esta investigación fue continuada por da Cunha et al. (2009) en co-autoría con el equipo del estudio anterior Lucena and Resende (2004), pero en este caso se propuso una variante de la implementación, utilizando un enfoque llamado "Lagrangian Non-Delayed Relax-and-Cut approach (NDRC)", técnica detallada en Lucena (2005). El principal objetivo es poder calcular una dualización lagrangeana con una cantidad exponencial de desigualdades. Su algoritmo es de carácter heurístico; utilizan el mismo modelo planteado en Lucena and Resende (2004), aplican *tests de reducción* que permiten descartar nodos y aristas que con seguridad no serán parte de una solución óptima y luego resuelven el problema utilizando las NDRC descriptas anteriormente. Su implementación fue ejecutada utilizando las mismas instancias que la anterior y se logran buenos tiempos de ejecución, llegando al óptimo en reiteradas ocasiones, sin sobrepasar el 12 % de gap respecto a la solución óptima.

En Klau et al. (2003) se introduce un algoritmo que consta de tres fases: (1) Una guía de pre-procesamiento que reduce el tamaño de la entrada sin afectar la estructura de la solución óptima. (2) Un algoritmo evolutivo estacionario que hace uso de una implementación exacta del problema para árboles. (3) Una rutina de pos-optimización resolviendo el problema con una relajación de un modelo de PLE que tiene como entrada la solución del paso anterior. En esta investigación logran reducir los tiempos de ejecución en instancias grandes,

[1] En el año 2013, los autores lanzaron una versión corregida de esta publicación

[2] Greedy Randomized Adaptive Search Procedure: es una metaheurística que combina procedimientos constructivos y de búsqueda local.

[3] generalized subtour elimination constraints (GSECs por sus siglas en inglés).

mostrando que la combinación de modelos exactos con algoritmos evolutivos puede llevar a soluciones que obtienen buenos resultados en corto tiempo.

Ljubić et al. (2004) presentó un modelo basado en una modificación del algoritmo presentado en Segev (1987), lo interesante de este algoritmo es que realiza una relajación al problema original, transformando el grafo original a un grafo dirigido, y agregando un nodo ficticio que además será el raíz en la solución. Este modelo tiene la particularidad de no generar una cantidad exponencial de restricciones. En el artículo no se hace mención a una implementación de este modelo.

***Definición* 2.1** *Dado un grafo* $G = (V, E)$ *diremos que el conjunto de los nodos* V *se puede particionar en dos subconjuntos*

$$P, yP^c, P \cup P^c = V, P \cap P^c = \varnothing.$$

***Definición* 2.2** *Se define como corte de un grafo al conjunto de los arcos incidentes hacia el exterior de* P.

$$(P, P^c) = (x, y), x \in P, y \in P^c, P \cup P^c = X, P \cap P^c = \varnothing$$

Posteriormente Ljubić et al. (2006) presentó un eficaz algoritmo Branch-and-Cut utilizando la misma transformación presentada en su artículo anterior (un grafo dirigido, con un nodo raíz agregado) el modelo presenta restricciones de desigualdad generadas por utilizar cortes (definición 2.2). Tener esta separación le permite utilizar el algoritmo de flujo máximo. En el artículo se menciona que su implementación llega al óptimo en todas las instancias.

Existe una variante del PCST, a menudo denominada "Quota and Budget" [4], esta variante ha sido tratada por Johnson et al. (2000)[5] quienes utilizando $quota = 0$, demostraron que existe un algoritmo de aproximación con factor 3 y de tiempo polinomial para este problema. Para su investigación utilizaron un algoritmo de aproximación para el "k-Minimum Spanning Tree k-MST".

[4]Esta es una generalización del *k-MST*. En primer lugar se busca el árbol con costo de aristas mínimo, que contenga vértices cuya penalización total es al menos un "valor" (*quota*) dado. Luego se busca el árbol con penalización máximo, dado que el coste total del aristas está dentro de un "presupuesto" (*budget*) determinado.

[5]En el mismo artículo que se presentó una variante del algoritmo expuesto enGoemans and Williamson (1995) pero con orden $O(n^2 log(n))$

Sobre la misma variante del problema Haouari and Siala (2006) desarrollaron un procedimiento de cotas inferiores basado en la descomposición de Lagrange, en conjunto con un algoritmo genético que fue probado en instancias generadas automáticamente obteniendo resultados muy cercanos al óptimo en gran porcentaje de ellos. Dos años después Haouari et al. (2008) introdujeron una versión generalizada del PCST que consiste en dividir el conjunto de nodos en k clústeres y definir un valor de quota para cada clúster. Para este problema, propusieron varias relajaciones lagrangianas para lograr el objetivo de obtener cotas inferiores al problema. En Haouari et al. (2010) presenta un algoritmo exacto para resolver el PCST, utilizando una la combinación de procedimientos de preprocesamiento, heurísticas y formulaciones programación lineal binaria. El algoritmo permite encontrar soluciones óptimas al PCST incluso para instancias grandes. Y finalmente los mismos autores continúan su investigación sobre esta generalización del PCST en Haouari et al. (2013), donde mencionan varias formulaciones de programación entera del PCST, entre ellas:

- **MTZ-based**: Proponen una variante de la formulación que presentaron Miller et al. (1960)[6], pero que es aplicable al PCST.

- **Pairwise precedence variables-based**: Proponen una variante de familia de formulaciones basadas en variables de precedencia aparejadas utilizadas en varias investigaciones[7] previas, pero también aplicable al PCST.

- **Flow-based**: Proponen la misma formulación que en Ljubić et al. (2004) realizando un análisis de cómo surge esta formulación, y las aplicaciones previas[8] que ha tenido.

En su investigación se realiza una comparativa entre los modelos propuestos tanto en la solución óptima como la relajación de cada uno de ellos.

En Biazzo et al. (2013) se propone un algoritmo estocástico[9] el cual es

[6]En esta investigación se propuso para la resolución del *Asymmetric Traveling Salesman Problem (ATSP)*

[7]Entre ellas: Gouveia and Pires (1999), Gouveia and Pires (2001), Sarin et al. (2005), Gouveia and Pesneau (2006), Sherali et al. (2006) también propuesta para resolver (ATSP)

[8]Multicommodity formulation, ha sido aplicada en varias oportunidades, entre ellas Beasley (1984) Sherali and Driscoll (2002), ambas investigaciones para la resolución del STP

[9]Éste está basado en el algoritmo *The Cavity Method at Zero Temperature* presentado en Mézard and Parisi (2003)

implementado utilizando el método del punto fijo[10], y además lo compara con algunos de los algoritmos del estado del arte como el de Goemans and Williamson (1995) y Ljubić et al. (2006), logrando buenos resultados a medida que crece el tamaño de las instancias.

También se han realizado numerosas investigaciones sobre las variantes del PCST. Entre ellas Könemann et al. (2013), en este artículo se detalla un algoritmo con factor $O(log|V|)$ para el *Prize-Collecting Node-Weighted Steiner Tree Problem (PCNWSTP)*. Su algoritmo tiene la propiedad de preservación de los multiplicadores de Lagrange, que puede ser útil en muchos contextos[11]. El algoritmo presentado también consta de dos fases; la primera de crecimiento y posteriormente de poda.

En Álvarez Miranda et al. (2013) se detalla una técnica que sigue los lineamientos de "Robust Optimizacion (RO)" ya que hace uso del algoritmo conocido como *B&S* propuesto por Bertsimas and Sim (2004). En el algoritmo B&S los parámetros están sujetos a un intervalo de incertidumbre, y el nivel de robustez es controlado mediante un parámetro de control. El objetivo de su algoritmo RO fue resolver el PCST y su versión budget y quota, para ello optaron por implementarlo utilizando un enfoque Branch-and-Cut. Su técnica fue probada en varias instancias, con diferentes niveles de incertidumbre, esta es una variante interesante ya que permite cierta flexibilidad en la solución óptima de acuerdo a un criterio previamente establecido.

En Akhmedov et al. (2016), se propone un algoritmo metaheurístico basado en la técnica de *Divide and Conquer (D&C)*[12]. El objetivo principal de su algoritmo es incrementar la aplicabilidad de los métodos exactos disponibles a grafos de gran tamaño, sin perder calidad en la solución. Su técnica se compone de una rutina de preprocesamiento, un algoritmo de agrupamiento (*clustering*) basado en metaheurística y posteriormente se aplica un método exacto para hallar el PCST en cada subgrafo generado producto de la aplicación de D&C.

Una de las publicaciones más recientes es la de Fischetti et al. (2016), donde se presenta un enfoque distinto a las formulaciones vistas anteriormente. En esta formulación no utiliza una variable de decisión asociada a las aristas,

[10] Algoritmo iterativo que permite resolver sistemas de ecuaciones, propuesto por Banach (1922)

[11] Ej: el k-MST o k-Steiner Tree

[12] Un algoritmo de D&C funciona dividiendo recursivamente un problema en dos o más sub-problemas del mismo tipo o de un tipo relacionado, hasta que estos se vuelven lo suficientemente simples para ser resueltos directamente. Cormen et al. (2001)

siendo necesario introducir el concepto de *nodo separado*. Esto hace que sea una formulación interesante de estudiar, aunque existen algunas consideraciones que deben tenerse en cuenta, dado que no es posible llevar el control sobre la ponderación de las aristas, se deben realizar ajustes en la penalidad de los nodos. Otro detalle a tener en cuenta, es que solamente funciona correctamente en grafos con ponderación de aristas moderadamente uniforme. Es posible que esto último sea absurdo pero lo cierto es que este modelo está pensado para ser utilizado como una opción para resolver el problema; este tema se volverá a tratar en las secciones 2.2.5 y 5.2.

En conclusión, en esta sección de investigación el objetivo es realizar un estudio comprensivo sobre los métodos exactos del PCST, fueron detalladas muy brevemente las formulaciones, los algoritmos, así como sus diferentes algunas de sus variantes para lograr resolver el problema. También se mencionaron algunos casos donde se introducen técnicas que logran resolver variantes del PCST.

En la siguiente sección se profundizará en algunas de estas formaciones, las cuales fueron seleccionadas por varios motivos; entre ellos por ser referentes a la hora de desarrollar el modelo presentado en el Capítulo 3, otras por su relevancia, así como por la cantidad de investigaciones que dieron a lugar por su carácter innovador o por la sencillez a la hora de ayudar a comprender el problema.

2.2. Métodos de resolución exactos

En esta sección se detallan cinco formulaciones exactas al problema, seleccionadas a partir del relevamiento de la sección anterior.

En cada caso se presentan respetando el siguiente esquema:

1. Una justificación de su elección.
2. Las variables de decisión que son utilizadas en el modelo.
3. El modelo, formulado como un PPL.
4. Análisis del modelo.
5. Conclusión.

2.2.1. Formulación 1

En 1995, Goemans and Williamson (1995) plantean el algoritmo de aproximación primal-dual para el r-PCST, que es una versión diferente del PCST clásico, donde se incluye un nodo r que es la raíz del árbol resultado (el nodo r siempre pertenecerá al conjunto V_T), lo relevante de su investigación fue construir un algoritmo de aproximación capaz de garantizar factor 2, del cual se darán más detalles en la Sección 2.3. Sumado a que es formulación es muy simple de comprender, constituyen los puntos claves de su inclusión en esta sección. Para el planteo en forma de PLE presentan las siguientes variables:

$$X_e = \begin{cases} 1 & \text{Si la arista } e \text{ es utilizada en el árbol solución} \\ 0 & \text{En otro caso} \end{cases}$$

para cada $e \in E$.

***Definición* 2.3** *Sea S el subconjunto de V que contiene exactamente a todos los vértices que no fueron recubiertos.*

Definimos como:

$$Z_N = \begin{cases} 1 & \text{Si } N \text{ coincide con } S \\ 0 & \text{En otro caso} \end{cases}$$

Para cada subconjunto de vértices $N \subset V$ sin el nodo r $(V \backslash r)$.

***Definición* 2.4** *Dado un conjunto V de vértices y un subconjunto S tal que $S \subseteq V$, definimos $\delta(S)$ como el conjunto de aristas que pertenecen al corte S, $(\delta(S) = \{(i,j) \in E : i \in S, j \in \overline{S}\})$*

Modelo PPLE planteado:

$$PCST_{G\&W} = \begin{cases} \text{Min:} \\ \sum\limits_{e \in E} c[e] * X_e + \left(\sum\limits_{N \subseteq (V \backslash r)} Z_N * \sum\limits_{i \in N} p(i) \right) \\ \text{Sujeto a:} \\ \sum\limits_{e \in \delta(S)} X_e + \sum\limits_{N \supseteq S} Z_n \geqslant 1 \qquad \forall S, N \subseteq (V \backslash r) \end{cases} \tag{2.1}$$

Tal cual se plantea en Goemans and Williamson (1995), en la función objetivo hay una restricción implícita producto de una relajación al problema, y establece que Z_N es no vacío en a lo sumo un subconjunto.

En la ecuación (2.1) se plantea que si un subconjunto S de vértices de V que no contienen la raíz ($S \in (V \backslash r)$), intersecta el subconjunto de vértices del árbol solución, el corte debe contener al menos una arista del árbol. De lo contrario, S estaría contenido dentro del conjunto de todos los vértices no cubiertos por el árbol.

Lo que hace tan breve este modelo, es el uso de la variable Z_N, con ello Goemans and Williamson logran abstraerse del concepto de "nodo seleccionado", restando únicamente aplicar la operación de sumatoria en toda la expresión para llegar al resultado. Sin embargo, como veremos en el capítulo siguiente utilizar el concepto de *subconjunto*, le lleva a tener problemas a la hora de la implementación.

2.2.2. Formulación 2

La que se plantea en Ljubić et al. (2006). En su investigación se introduce una relajación al problema original para grafos no dirigidos, resultando en un problema de grafos dirigidos. Luego de publicada esta investigación Ljubić et al. (2006) hicieron público un software[13] para la resolución exacta del PCST. Sirvió para el estudio de casos reales[14] y a su vez fue utilizado como referencia en variadas investigaciones sobre el mismo problema[15]. Estos detalles fueron los pilares fundamentales de incluir esta formulación, como una de las seleccionadas.

Partiendo de un grafo $G = (V, E)$ no dirigido, es necesario definir un nuevo conjunto de variables $G_{SA} = (V_{SA}, A_{SA})$ para poder obtener un grafo dirigido, las variables son:

- Un conjunto $V_{SA} = V \cup \{r\}$ que incluya los vértices del grafo y el nodo raíz.
- Un conjunto $R_{SA} = \{v \in V |\ p(v) > 0\}$
- Un conjunto A_{SA} que contenga las dos aristas (i, j) y (j, i) por cada $(i, j) \in E$ más un conjunto de aristas del nodo raíz r, con destino los vértices del grafo que tengan penalización positiva únicamente.
- Un vector c' de costos definidos para cada arista de A_{SA} con valores distintos dependiendo del nodo origen de la arista; si la arista es (r, j) el costo para esa arista será $-p_j$ mientras que en las aristas $\{(i, j) : i \neq j\}$ tienen costo $c_{ij} - p_j$

$$X_e = \begin{cases} 1 & \text{Si la arista e es utilizada en el árbol solución} \\ 0 & \text{En otro caso} \end{cases}$$

Para toda arista e en A_{SA}

$$y_v = \begin{cases} 1 & \text{Si el vértice v es utilizado en el arbol solución} \\ 0 & \text{En otro caso} \end{cases}$$

[13] El software llamado dhea, fue desarrollado en conjunto con otros colaboradores de la Universidad de Viena, fue implementado utilizando CPLEX y sus principales características son la presentada en esta formulación

[14] Tuncbag et al. (2012), y también en Dittrich et al. (2008)

[15] Biazzo et al. (2013)

Para todo vértice v en $V_{SA}\backslash\{r\}$

***Definición* 2.5** *Dado un conjunto V de vértices, un subconjunto S tal que $S \subseteq V$ y su complemento $\overline{S} = V_{SA}\backslash S$, definimos $\delta^-(S) = \{(i,j)|i \in \overline{S}, j \in S\}$.*

***Definición* 2.6** *Dado un conjunto A de aristas tal que $A \subset A_{SA}$, definimos $x(A) = \sum_{e \in A} X_e$ para cualquier subconjunto de aristas.*

A continuación se presenta el modelo como un PPLE,

$$
PCST_{Ljubić} = \begin{cases}
\text{Min:} & & \\
\sum\limits_{ij \in A_{SA}} c'_{ij} * X_{ij} + \sum\limits_{i \in V_{SA}} p(i) & & \\
\text{Sujeto a:} & & \\
\sum\limits_{ji \in A_{SA}} X_{ji} = y_i & \forall i \in (A_{SA}\backslash r) & (2.2) \\
x(\delta^-(S)) \geqslant y_k & k \in S, r \notin S, \forall \subset V_{SA} & (2.3) \\
\sum\limits_{ri \in A_{SA}} X_{ri} = 1 & & (2.4) \\
X_{rj} \leqslant 1 - y_i & \forall i < j, i \in R_{SA} & (2.5) \\
\sum\limits_{ji \in A_{SA}} X_{ji} \geqslant \sum\limits_{ij \in A_{SA}} X_{ij} & \forall i \notin R_{SA}, i \neq r & (2.6) \\
X_{ij}, y_i \in \{0,1\} & \forall (i,j) \in A_{SA}, \forall i \in (V_{SA}\backslash\{r\}) & (2.7)
\end{cases}
$$

Los aspectos a destacar de esta formulación son varios; la ecuación (2.3) recibe el nombre de *desigualdad de conectividad*, la cual permite garantizar un camino de r a todo vértice v presente en el árbol solución. La ecuación (2.5) es llamada restricción de *a-simetría*, es para lograr que la solución cumpla con característica particular de árboles. La ecuación (2.6) tiene como objetivo regular el balance de flujo, en todos los casos la arista incidente debe ser igual a la aristas salientes. Varios de estos conceptos se tratan con profundidad de detalles en el Capítulo 3 con el abordaje de la formulación exacta planteada en esta investigación.

2.2.3. Formulación 3

Esta formulación fue extraída de Ljubić et al. (2004), es una variante de la formulación anterior por lo que utiliza las mismas estructuras. En el artículo se menciona que Segev (1987) utilizó una formulación muy similar para el *Single Vertex-Weighted Steiner tree problem*, y como se mostró en la Sección 2.1, el *Multy-Commodity Flow Formulation (MCF)* (generalización del anterior) también fue utilizado en Beasley (1984) Sherali and Driscoll (2002) para resolver el STP.

En este caso esta formulación es introducida bajo el nombre *Single-Commodity Flow Formulation (SF)* y esta es uno de los modelos más simples.

Además de ser una formulación ampliamente estudiada, fue seleccionada porque es una de las pocas relevadas que no utilizan subconjuntos, para lidiar con la generación de ciclos.

El grafo $G = (V, A, c, p)$ se modifica a un grafo $G_{SA} = (V_{SA}, A_{SA}, c_{ij}, p)$, con aristas dirigidas que cumplan lo siguiente:

- Un conjunto $V_{SA} = V \cup \{r\}$ que incluya los vértices del grafo y el nodo raíz.
- Un conjunto $R_{SA} = \{v \in V | p(v) > 0\}$
- Un conjunto $A_{SA} = \{(i,j)(j,i) | (i,j) \in A\} \cup \{(r,k) | \forall k \in R_{SA}\}$
- Un conjunto para almacenar los costos de las aristas definido como:

$$c''_{ij} = \begin{cases} c'_{ij} - p'_j & \forall i,j \in A + SAi \neq r \\ -p'_j & \forall (r,j) \in A_{SA} \end{cases} \tag{2.1}$$

Un subgrafo T_{SA} de G_{SA}, tal que forma un árbol dirigido con raíz r, es un árbol de Steiner. Es fácil ver que tal subgrafo corresponde a una solución del PCST, si r tiene grado 1 en G_{SA} (árbol factible). Y en particular, un árbol factible con un costo total de aristas mínimo, es un PCST óptimo.
El problema de encontrar un T_{SA} mínimo se modela como un PPLE, para ello se utilizarán las mismas variables del caso anterior:

$$X_e = \begin{cases} 1 & \text{Si la arista e es utilizada en el árbol solución} \\ 0 & \text{En otro caso} \end{cases}$$

Para toda arista en A_{SA}

$$y_v = \begin{cases} 1 & \text{Si el vértice v es utilizado en el aŕbol solución} \\ 0 & \text{En otro caso} \end{cases}$$

Para todo vértice v en $V_{SA}\backslash\{r\}$

La formulación resultante puede escribirse como sigue:

$$PCST_{SF} = \begin{cases} \text{Min:} & & \\ \sum_{ij \in A_{SA}} c''_{ij} * X_{ij} + \sum_{i \in V_{SA}} p(i) & & \\ \text{Sujeto a:} & & \\ \sum_{ji \in A_{SA}} X_{ji} = y_i & \forall i \in (V_{SA}\backslash r) & (2.8) \\ \sum_{ji \in A_{SA}} f_{ji} - \sum_{iji \in A_{SA}} f_{ij} = y_i & \forall i \in R_{SA} & (2.9) \\ \sum_{ji \in A_{SA}} f_{ji} - \sum_{iji \in A_{SA}} f_{ij} = 0 & \forall i \in V_{SA}\backslash R_{SA}, i \neq r & (2.10) \\ 0 \leqslant f_{ij} \leqslant (|V_{SA}| - 1) & \forall (i,j) \in A_{SA} & (2.11) \\ \sum_{ri \in A_{SA}} X_{ri} = 1 & & (2.12) \\ X_{ij}, y_i \in \{0,1\} & \forall (i,j) \in A_{SA}, \forall i \in (V_{SA}\backslash\{r\}) & (2.13) \end{cases}$$

El término constante en la función objetivo se añade de manera que (SF) proporcione el valor deseado de la solución global. La ecuación (2.8), garantiza que cada vértice seleccionado tiene exactamente un predecesor en su trayectoria desde la raíz. Las restricciones clásicas de preservación de flujo están dadas por (2.9) y (2.10), con la particularidad de que la primera requiere que cada vértice seleccionado reciba una unidad de flujo que se consumirá en este vértice. La restricción (2.11) fuerza a que los arcos que son utilizados por cualquier flujo sean incluidos en el árbol dirigido final. Por último, la restricción (2.13) asegura que la raíz artificial r está conectada sólo a un único vértice lo cual es crucial para la conexión de la solución.

La simplicidad de esta formulación viene dada por el hecho de imponer al árbol un nodo raíz "artificial", y la introducción de flujos (mediante el uso de la variable $f_{i,j}$ que representan la cantidad de flujo que pasa sobre el arco (i,j)) para definir la estructura del árbol. La esencia del modelo, es enviar una unidad del flujo desde el vértice raíz r, a cada vértice del cliente en el árbol de solución. Finalmente son los valores de $x_{i,j}$ indican las trayectorias dirigidas desde r a cada vértice seleccionado, y esto dependerá de si el nodo ha sido seleccionado por la variable y_i.

2.2.4. Formulación 4

Una formulación muy clara y ágil de leer es la que plantea Lucena and Resende (2004), aunque lo interesante de su investigación es el algoritmo de planos de corte que presentan para obtener cotas inferiores del problema, logrando muy buenos resultados, su investigación sirvió como punto de partida para varias investigaciones como es el caso de da Cunha et al. (2009) y Álvarez Miranda et al. (2013).

Además de ser fácil de analizar dada su simplicidad, el porqué de incluir esta formulación, tiene fundamento en que será una referencia a comparar lo mencionado sobre la obtención de cotas inferiores al PCST, esta comparativa se realizará en la Sección 6.4.2.

Dado un grafo $G(V, E, c, p)$, proponen el uso de las siguientes variables:

- Un sub conjunto $S \subseteq V$ de vértices.
- Un conjunto E de aristas.
- un vector que contenga los valores c_e de costos definidos para cada $e \in E$.
- un vector que contenga los valores p_v de penalización definidos para cada $v \in V$.

El uso de dos variables de decisión:

- Para cada arista una variable $x_e \in \{0, 1\}$.
- Para cada vértice una variable $y_v \in \{0, 1\}$.

Y por último considerar siguientes denominaciones

- $E(S) \subseteq E$ al conjunto de aristas con ambos extremos en S,
- $x(E(S))$ al valor de la suma $\sum_{e \in E(S)} x_e$.
- $y(S)S \subseteq V$ a la suma $\sum_{s \in S} y_s$

$$
PCST_{LyR} = \begin{cases}
\text{Min:} & & \\
\sum_{e \in E} c_e x_e + \sum_{v \in V} p_v(1 - y_v) & & \\
\text{Sujeto a:} & & \\
x(E) = y(V) - 1 & & (2.14) \\
x(E(S)) \leqslant y(S \backslash \{j\}) & \forall j \in S, S \subseteq V, & (2.15) \\
0 \leqslant x_e \leqslant 1 & \forall e \in E & (2.16) \\
0 \leqslant y_i \leqslant 1 & \forall i \in V & (2.17)
\end{cases}
$$

El cometido de la restricción (2.14) es que cumpla una de las condiciones necesarias para que un grafo sea árbol, que incluye las cardinalidades de vértices y aristas. La restricción (2.15) garantiza que la solución no tenga ciclos

(esta restricción recibe el nombre de *restricción generalizada de eliminación de sub-caminos, GESC por sus siglas en inglés*, fue mencionada por primera vez en Dantzig et al. (1954)). Finalmente las restricciones (2.16) y (2.17) validan las cotas superior e inferior de cada variable.

Esta investigación también es de las primeras que surgieron respecto al PCST, y es necesario subrayar que al igual que las primeras dos, utiliza los subconjuntos de un conjunto para definir la región factible.
En la Sección 6.4.2 se realiza una breve comparativa de las relajaciones de los modelos propuestos en esta investigación.

2.2.5. Formulación 5

Esta formulación es presentada en Fischetti et al. (2016) y cuenta con un enfoque distinto a las anteriores; tiene como único objetivo resolver las instancias cuyo costo en las aristas es moderadamente uniforme.

El aspecto innovador de esta formulación, es que no presenta una variable de decisión asociada al conjunto de aristas. Esta es la razón fundamental por la que fue seleccionada. En tal caso, el problema que se está resolviendo es el "Maximum-Weight Connected Subgraph Problem (MWCSP)", aunque es posible realizar una variante a los datos de entrada para poder resolver el PCST.

Este modelo utiliza el concepto de *separador de nodos*: Dados dos nodos distintos k y l de V, llamaremos a un subconjunto de nodos N incluido en $V\backslash\{k,l\}$ como un $(k,l)-separador$ si y solo si luego de eliminar N del conjunto original V, no hay ningún camino entre los nodos (k,l) en el grafo G.

Un separador N es *minimal*, si $N\backslash\{i\}$ no es un $(k,l)-separador$ para cualquier $i \in N$.

A la familia de todos los $(k,l)-separadores$ se denotará como $\mathcal{N}(l,k)$.

Una observación no menor es la siguiente: Para asegurarse que un subconjunto de nodos elegidos está conectado, es suficiente con imponer conectividad entre pares de terminales, (por efecto de la minimización presente en la función objetivo). En consecuencia solamente interesan los separadores entre pares de terminales.

Se asume que el grafo $G = (V, E, C, P)$ cumple con tener: penalizaciones p_v asociadas a cada nodo $v \in V$ y costos c_e iguales para cada arista $e \in E$

Para poder describir correctamente un modelo basado únicamente en no-

dos, se trabajará con la variable $\tilde{c}_v$ definida en función de los costos de aristas y la penalización de los nodos siguiendo el siguiente esquema:

$$\tilde{c}_v = c - p_v \forall v \in V$$

Se considera un conjunto de terminales T_r que eventualmente puede ser vacío, y en tal caso es el PCST que se ha tratado en este documento.

Se define un conjunto denominado *terminales potenciales* de la siguiente forma: $T_p = \{v \in V \setminus T_r \mid \exists\{u, v\}\ sujeto\ a\ c_{uv} < p_v\}$ y el conjunto $T = T_p \cup T_r$.

y finalmente se define el conjunto $T = T_p \cup T_r$.

Asimismo se define una única variable de decisión:

$$y_v = \begin{cases} 1 & \text{Si el nodo v es seleccionado} \\ 0 & \text{En otro caso} \end{cases}$$

El modelo basado en nodos queda definido de la siguiente forma:

$$PCST_{Fischetti} = \begin{cases} \text{Min:} & & \\ \sum_{v \in V} \tilde{c}_v y_v + (\sum_{v \in V} p_v) - c & & \\ \text{Sujeto a:} & & \\ y(N) \geqslant y_i + y_j - 1 & \forall i, j \in T, i \neq j, \forall N \in \mathcal{N}(i, j) & (2.18) \\ y_v = 1 & \forall v \in T_r & (2.19) \\ y_v \in \{0, 1\} & \forall v \in V \setminus T_r & (2.20) \end{cases}$$

con $y(N) = \sum_{v \in N} y_v$

La restricción (2.18) es para asegurar conectividad de la solución; cuando dos nodos terminales i, j distintos sean parte de la solución, a lo sumo un nodo de los separadores $N \in \mathcal{N}(i, j)$ debe ser seleccionado para poder asegurar un camino entre i y j.

En Fischetti et al. (2016) se menciona que este modelo tal cual fue presentado, es una versión básica y para poder llegar a buenos resultados es necesario realizar un posprocesamiento de la solución con dos algoritmos más, llamados "node-degree inequalities" y "2-cycle inequalities".

Existen dos razones para incluirlo como una de las formulaciones. La primera intenta mostrar la creatividad a la hora de resolver un problema de este estilo, Backes et al. (2012) fueron los primeros en proponer un modelo basado

solamente en nodos, allí se menciona la baja cantidad de restricciones generadas, respecto al modelo clásico. La otra razón de incluirlo se basa en que este modelo es utilizado para algunas instancias en la implementación que se detalla en la Sección 5.2.

2.3. Métodos de resolución aproximados

Las técnicas de resolución aproximada son la opción ideal cuando se está en busca de resolver el problema de forma rápida, a costa de no obtener el mejor resultado.

En referencia a este problema, en 1995 Goemans and Williamson presentaron lo que llamaron "*General approximation technique for a large class of graph problems*"; una técnica capaz de producir algoritmos de aproximación con factor 2 para una variada serie de problemas, y lo más destacable es que su ejecución es de orden $O(n^2 log(n))$ (en referencia a esta investigación, este orden se aplica para el r-PCST, mencionado anteriormente, y para la versión sin nodo raíz, el orden es $n * n^2 log(n)$). Su trabajo dio lugar a numerosas investigaciones, algunas de ellas mencionadas en la sección Sección 2.1.

La técnica puede aplicarse para resolver a una variada cantidad de problemas, en particular es aplicable a los problemas de Steiner.

Como se menciona en Bauza and Olivera (2017) El algoritmo además de ser muy eficiente, es muy simple; consta de tres grandes fases, a continuación se enuncia muy brevemente en qué consiste cada una de ellas:

1. **Inicialización**: Consiste en inicializar las estructuras necesarias para la ejecución del algoritmo.

2. **Crecimiento**: Los nodos están todos desconectados (bosque), dado que aún no existen aristas agregadas, y en esta fase se comienzan a agregar.

3. **Poda**: El objetivo es encontrar y descartar aristas redundantes seleccionadas en la fase anterior, dando como resultado el árbol buscado.

Existe una alternativa a la última fase, propuesta en Johnson et al. (2000) llamada *strong pruning*, estrategia que respeta el esquema de búsqueda en profundidad[16], descartando los nodos cuya penalización es menor al costo de

[16]En ingles Depth-first search (DFS), es un algoritmo recursivo que partiendo de un nodo raíz recorre todas las ramas llegando hasta las hojas. La lógica del algoritmo se ejecuta al retornar de la llamada recursiva

llegar hasta él.

También en Johnson et al. (2000) se enuncia el siguiente teorema:

Teorema 1 (Teorema comparación GW-pruning y strong pruning) *Sea G un grafo con nodo raíz en un vértice v_0, y sea T un sub-árbol cualquiera de G, Sea T' un sub-árbol cualquiera que contiene al vértice v_0, y sea T_{SP} el árbol generado de a partir de T aplicando la técnica de strong pruning. Se cumple que $GW(T_{SP}) \leqslant GW(T')$*

Con este teorema, se puede seguir garantizando el factor 2 del algoritmo original.

Bauza and Olivera (2017) realizaron la implementación de este algoritmo para el PCST, utilizando la variante strong pruning aplicada al algoritmo de Goemans and Williamson para el PCST. En la sección Sección 5.3 se mencionan algunos detalles de esta implementación, y en la Sección 6.6 se comparan los tiempos de ejecución y la proximidad al óptimo entre su implementación utilizando algoritmos aproximados y las implementaciones de algoritmos exactos propias de esta presente investigación.

Capítulo 3

Detalles de Formulación Exacta

Como se puede apreciar en las formulaciones descriptas en el capítulo anterior, y es una particularidad general de la literatura sobre el PCST, la mayoría de los modelos propuestos presentan restricciones con subconjuntos, esto lo hace muy sencillo de interpretar, pero también lo hace muy difícil de implementar.

Cuando se está ante una restricción con: "... $\forall S \in V$ ...", se debe calcular el conjunto de *partes*[1] de V, cuya cardinalidad depende exponencialmente de la cantidad de elementos del conjunto inicial (en nuestro caso si $|V| = n \Rightarrow |\mathscr{P}(V)| = 2^n$). Una restricción tan simple teóricamente como es la (2.1) de la formulación 1 (Subsección 2.2.1), a medida que crece n, también lo hace la cantidad de instancias para esa misma restricción y de forma exponencial, lo que acaba con la esperanza de conseguir una buena performance en cuanto a tiempos de ejecución refiere.

3.1. Formulación propuesta

En esta sección se detalla la formulación del modelo propuesto para la solución exacta del PCST. Tal como se adelantara en la introducción, éste es uno de los pilares fundamentales de esta investigación.

Recapitulando, el enunciado del problema es el siguiente:
Dado un grafo $G = (V, E, C, P)$, el objetivo es encontrar un árbol $T =$

[1]Partes de un conjunto: Dado un conjunto S de tamaño n, el conjunto *partes* de S denotado como $\mathscr{P}(s)$, es el conjunto de todos los subconjuntos de S. El orden de este conjunto es 2^n Vardi (1991).

$(V_t, E_t) \in G, V_T \in v, E_T \in$ E que minimice la siguiente expresión:

$$costo(T) = \sum_{v \notin V_T} p(v) + \sum_{(i,j) \in E_T} c(i,j)$$

Para la posterior implementación de este modelo, fue necesario pensar el problema desde un enfoque diferente a como se plantea en la mayoría de los modelos ya existentes. El objetivo era evitar el uso de subconjuntos, por su complejidad inherente.

3.2. Referencias para el modelo planteado

El modelo está inspirado en las ecuaciones de circuitos de Kirchhoff[2]. En la investigación Bonchev et al. (1994) realizada por especialistas en matemáticas aplicadas a la química, sobre los ciclos presentes a nivel molecular, introdujeron en su análisis el concepto de *índice de Kirchhoff* para referirse a la *distancia de resistencia* en un grafo, haciendo un paralelismo con el estudio que años atrás había realizado el propio Kirchhoff sobre la ley de conservación de carga en circuitos eléctricos.

Los investigadores Bonchev et al. notaron que para grafos conexos, la distancia de resistencia y la distancia usual entre todo par de nodos de un grafo, sufría variaciones conforme existían o no ciclos; fijados dos nodos de un grafo que presentaba ciclos, la distancia usual era mayor o igual a la distancia de resistencia, mientras que en grafos acíclicos (árboles) la distancia era siempre igual. Se pueden encontrar más detalles sobre una investigación posterior en Xiao and Gutman (2003).

Las siguientes definiciones, fueron extraídas de Romero (2009), y sirven para comprender mejor los conceptos que se introducen luego:

Definición* 3.1** *Una* ***red *es un grafo dirigido ponderado* $G = (V, E, c)$ *donde* $c : E \to R^+$ *es una función que asocia a cada enlace una capacidad.*

Definición* 3.2** *Una* ***red s-t *es una red donde además se identifican dos terminales distinguidos: la fuente. denotada como* s*, y el pozo o sumidero, denotado como* t*.*

[2]La ley propuesta en 1845, establece el principio de conservación de la carga eléctrica: En cualquier nodo (unión) en un circuito eléctrico, la suma de las corrientes que entran hacia ese nodo es igual a la suma de las corrientes que salen de ese nodo Clayton (2001).

Teniendo en cuenta la notación habitual donde $I^+(x)$ refiere al conjunto de enlaces salientes desde el nodo x, e $I^-(x)$ el conjunto de enlaces entrantes a x se define lo siguiente:

Definición* 3.3** *Dada una red r-s, un* ***flujo *de s a t es una función $\varphi : E \to R^+ \cup \{0\}$ talque :*

1. $\varphi(e) \leqslant c(e), \forall e \in E,$
2. $\sum_{e \in I^-(s)} \varphi(e) = \sum_{e \in I^+(t)} \varphi(e) = 0,$
3. $\sum_{e \in I^-(s)} \varphi(e) = \sum_{e \in I^+(t)} \varphi(e), \forall x \neq s, x \neq t.$

La condición 1 indica que el flujo no puede superar la capacidad, la condición 2 es la definición de fuente s y pozo t, la condición 3 es la ley conservación del flujo, o ley de Kirchhoff.

Estructuras

En este caso, al igual que en muchas de las formulaciones existentes sobre problemas de Steiner, es necesario convertir el grafo original $G = (V, E, C, P)$ en un grafo subyacente dirigido $G_c = (V_c, E_c, C_c, P_c)$ tal que;

- $V_c = \{v \mid v \in V\}^*$
- $E_c = \{(i, j),\ (j, i) \mid \forall (i, j) \in E\}$
- $C_c = \{(i, j, c),\ (j, i, c) \mid \forall (i, j, c) \in C\}$
- $P_c = \{(v, p) \mid (v, p) \in P\}^*$

* estos conjuntos no sufren modificaciones respecto del original.

Variables de decisión

El modelo hace uso de las siguientes variables de decisión:
Variable que lleva el control de los nodos seleccionados

$$x_i = \begin{cases} 1 & \text{Si el nodo } i \text{ está en la solución} \\ 0 & \text{sino} \end{cases}$$

Variable que lleva el control de las aristas seleccionadas

$$X_{(i,j)} = \begin{cases} 1 & \text{Si la arista } (i,j) \in E_c \text{ está en la solución} \\ 0 & \text{sino} \end{cases}$$

Variable que lleva el control de los caminos seleccionados

$$Y_{(i,j)}^{u,v} = \begin{cases} 1 & \text{Si la arista } (i,j) \in E_c \text{ es utilizada en un camino} \\ & \text{de u hacia v en el sentido } u \to i \to j \to v \\ 0 & \text{sino} \end{cases}$$

El modelo

El modelo planteado como un problema de programación lineal entera, es el siguiente:

$$PCST = \begin{cases} \text{Min:} & & \\ \sum_{v \in V_c} p(v) * (1 - x_v) + \sum_{(i,j) \in E_c} c(i,j) * X_{(i,j)} & & \\ \text{Sujeto a:} & & \\ \sum_{(u,i) \in E_c} Y_{(u,i)}^{u,v} \geqslant x_u + x_v - 1 & \forall u, v \in V_c & (3.1) \\ \sum_{(j,v) \in E_c} Y_{(j,v)}^{u,v} \geqslant x_u + x_v - 1 & \forall u, v \in V_c & (3.2) \\ \sum_{(i,p) \in E_c} Y_{(i,p)}^{u,v} - \sum_{(p,j) \in E_c} Y_{(p,j)}^{u,v} \geqslant x_u + x_v - 2 & \forall u, v \in V_c, p \in (V_c \backslash \{u, v\}) & (3.3) \\ Y_{(i,j)}^{u,v} + Y_{(j,i)}^{u,v} \leqslant X_{(i,j)} + X_{(j,i)} & \forall (i,j) \in E_c, \forall u, v \in V_c & (3.4) \\ 2 * X_{(i,j)} \leqslant x_i + x_j & \forall (i,j) \in E_c & (3.5) \\ X_{(i,j)} + X_{(j,i)} \leqslant 1 & \forall (i,j) \in E_c & (3.6) \\ X_{(i,j)}, x_p, Y_{(i,j)}^{u,v} \in \{0,1\} & \forall p, u, v \in V_c, (i,j) \in E_c & (3.7) \end{cases}$$

Variable Y

En este punto, es necesario hacer un contraste entre las definiciones de este comienzo del capítulo y el modelo planteado. La variable $Y_{(i,j)}^{u,v}$ es la que utiliza estos conceptos, sin embargo es necesario aclarar que respecto a la definición 3.3 lo hace únicamente con flujos de una unidad identificados como *pulsos*.

Además los conceptos de terminal y fuente de la definición 3.2, no están presentes tal y como se detalló en la definición, sino que se modelaron asociados a un *camino* que los interconecta. Esto quiere decir que por cada camino, existe únicamente un nodo fuente y un nodo pozo.

A modo de observación, en el grafo resultado hay exactamente la misma cantidad de nodos fuente y pozo como caminos seleccionados.

Existen varias investigaciones que utilizan la misma lógica como por ejemplo en Parodi (2011) donde se utiliza una variable de decisión similar de cuatro dimensiones para resolver una red MPLS utilizando también un esquema de PLE.

3.3. Comentarios sobre la formulación

En este modelo es intuitivo ver que rol cumple cada una de las restricciones. Fijados los vértice u y v, si están seleccionados se debe cumplir:

- (3.1) Que exista un camino en dirección $u \rightarrow v$, para ello es necesario que entre todas las aristas dirigidas con ***salida*** en u, exista alguna con extremo en algún nodo i (adyacente a u) seleccionada, esto se modela con el concepto de camino utilizando la variable $Y_{(u,i)}^{u,v}$
- (3.2) Ídem que la anterior, pero maneja el concepto de ***llegada*** a v, mediante un nodo intermedio j (adyacente a v).
- (3.3) Para todo nodo intermedio que interviene en ese camino, debe existir una única arista de entrada proveniente desde un nodo i, así como una única arista de salida hacia un nodo j, con el objetivo de cancelar el ***balance***.

Estas tres decisiones garantizan un camino entre un par de nodos (que el modelo decidió que estén en la solución), enviando un pulso desde la arista de salida en dirección a la arista de llegada y controlando que el balance sea cero en todo punto intermedio.

Las restricciones siguientes tienen el cometido de convertir nuevamente el grafo subyacente, a un grafo no dirigido:

- (3.4) La arista (i, j) en el grafo sin dirección debe estar seleccionada solamente si quedó seleccionada en alguna de las dos direcciones para conectar u con v.
- (3.5) Si la arista (i, j) está en la solución, los nodos i y j tienen que estar.
- (3.6) No se debe seleccionar a la vez una arista (i, j) y su homóloga en sentido contrario (j, i).

3.4. Segunda implementación de control

Durante el proceso de implementación, se hizo indispensable contar con un segundo modelo que resuelva el problema. Al comienzo el objetivo era comprobar si efectivamente el nuevo modelo de solución exacta (propuesto en la Sección 3.2) se llegaba a la solución óptima. Posteriormente, en la etapa de análisis de ejecución, esta implementación serviría de referencia para comparar los tiempos necesarios para encontrar la solución óptima.

De los modelos detallados en el Capítulo 2, finalmente la decisión fue implementar el tercer modelo, presentado en la sección Subsección 2.2.3. La particularidad de la formulación 3, es que genera una cantidad polinomial de restricciones, utilizando el concepto de flujos salientes desde un nodo raíz agregado artificialmente.

Una vez más, cabe reiterar lo mencionado en los primeros párrafos de este capítulo sobre la cantidad de restricciones que se generan dependiendo de la lógica que utilice el modelo. Esto le da sentido a esta decisión, ya teniendo en mente la etapa de implementación del modelo.

Capítulo 4

Detalles sobre las implementaciones

En este capítulo se listan los detalles sobre la implementación de los modelos descriptos en la sección anterior. Además se mencionan algunos detalles sobre los datos de entrada, los datos de salida y su posterior visualización.

Herramientas a utilizar

En el Capítulo 1 se mostraron algunos técnicas de resolución de problemas *NP-Difícil* utilizando métodos exactos, y el el Capítulo 2 se mostraron algunas investigaciones detallando la técnica de la resolución exacta utilizada para el PCST.

En este caso, luego de evaluar seriamente todas las posibilidades disponibles, se optó por resolver el problema utilizando la API de CPLEX; un software cuyo principal objetivo es resolver problemas de optimización utilizando una combinación eficiente de las técnicas existentes, para lograr el mejor tiempo posible a la hora de hallar la solución. Está implementado utilizando el lenguaje C, que no utiliza *garbage collector* ni otras rutinas que se ejecutan en paralelo, las cuales están disponibles en otros lenguajes. Además es un lenguaje con gran performance en lo que refiere a programación numérica.

CPLEX está disponible desde 1988 sin embargo los mayores avances sobre el motor que utiliza para la resolución de problemas se dieron a partir del año 2009 cuando fue adquirido por la compañía IBM. Estos avances refieren a técnicas de metaheurísticas para MIP, así como la versión de 64 bits multiplataforma.

CPLEX es solamente un motor de cálculo, cuya única entrada es un modelo

de programación lineal, el cual ya ha sido preprocesado e ingresado por software externo, mediante llamadas a la API, y una llamada final para resolver la solución.

Ya decidida la herramienta de optimización a utilizar, el paso siguiente fue decidir cómo se harían las llamadas a CPLEX.

Finalmente la decisión fue utilizar la interfaz gráfica de OPL llamada *OPLIDE*: un plugin de eclipse que permite realizar el preprocesamiento de los datos, mediante la interfaz de código que provee. Además permite visualizar los datos en forma ordenada.

OPL está optimizado pura y exclusivamente para los problemas de optimización. Además viene como un software adicional en el paquete de instalación provisto por IBM.

En este punto corresponde abrir paréntesis y comentar que durante el desarrollo de esta investigación se logró un intercambio muy fructífero con la Dra. Ljubić (quien creó la *formulación 2* y además realizó las modificaciones correspondientes al modelo *Single-Flow* clásico, transformándolo en lo que se presentó como la *formulación 3*, ambas descriptas en el Capítulo 2) vía correo electrónico, ella facilitó un pre-print de su última investigación , así como información sobre un evento[1] de implementación. En la página del evento es posible encontrar un gran catálogo con juegos de datos referentes a instancias reales para alimentar el modelo, además se pueden visualizar los tiempos de ejecución de los participantes[2]. Sin embargo lo más destacable fue encontrar el estándar STP, que apareció en el momento exacto donde se hacía necesario contar con un lenguaje común para los datos de entrada, a la hora de ejecutar el software.

Un detalle más, la Dra. Ivana Ljubić integra un equipo de investigadores quienes se presentaron a la competencia con dos implementaciones para

[1] 11th DIMACS Implementation Challenge: estuvo dedicado al estudio de los problemas de árboles de Steiner, de forma práctica y teórica. Los desafíos de implementación del instituto DIMACS son muy esporádicos, aunque suelen reunir a las grandes personalidades del ambiente matemático.

[2] http://dimacs11.zib.de/contest/results/results.html
Las ejecuciones se realizaron en un cluster de Zuse Institute Berlin con las siguiente especificaciones:

- 32 compute nodes.
- Intel Xeon X5672 3.20GHz (8 cores).
- 48 GB RAM.

resolver los problemas propuestos; mozartballs (solución exacta) y staynerd[3] (solución aproximada utilizando heurísticas), ambos fueron ganadores en sus respectivas áreas, logrando muy buenos tiempos de resolución en todas las instancias y ganando la mayoría de los desafíos referentes al PCST.

Dificultades en la implementación

Ya aclarados todos los aspectos previos, a continuación se comentan los detalles referentes el proceso de implementación del modelo, que tomó varias semanas de codificación y algunas otras de testeo y documentación. Durante este proceso el modelo original fue modificado hasta llegar a una versión estable y funcional.

OPLIDE es muy sencillo e intuitivo, está pensado para personas con perfil académico-matemático lo que hace muy sencillo su uso, aunque es probable que a veces desoriente a quien tenga un perfil más técnico.

Notación

Antes de entrar en detalle referente a la implementación, dado que se lograron implementar dos modelos exactos, que cumplen con hallar la solución óptima al problema, proponer como denominación:

- **Modelo de caminos**: Al modelo propio, detallado al comienzo del Capítulo 3
- **Modelo de flujos**: Al modelo presentado como formulación 3 en el Capítulo 2

4.1. Datos de entrada

En un comienzo no estaba claro el formato a utilizar para la entrada de datos, una posibilidad era utilizar un formato común estilo matriz de adyacencia. Finalmente la decisión se hizo trivial cuando surgió la posibilidad de leer los datos en formato STP. Debido a que en el sitio web del DIMACS Challenge, estaba disponible un enorme catálogo con datos e instancias de redes reales, y otras creadas automáticamente, y eso terminó de inclinar la balanza en favor del ya mencionado formato.

[3]Un juego de palabras con el apellido de quien diera nombre a esta problemática.

Formato STP: Steiner Tree Problem.

Como su nombre lo sugiere STP es un formato pensado desde el comienzo, para trabajar exclusivamente con problemas de árboles de Steiner Koch et al. (2001).

Un aspecto interesante de este formato; abarca todas las posibles combinaciones que existen para las variantes del STP.

El estándar cuenta con documentación web muy bien detallada, aunque en esta investigación se hizo uso solamente de una pequeña parte de las opciones disponibles que refieren al PCST. En el apéndice 1 es posible ver detalles del estándar STP.

4.2. Planteo

Una vez decidido el formato de entrada, es necesario contar con un *módulo de parseo*, que se encargue de leer el archivo STP, y realizar la tarea de almacenar en memoria las estructuras de datos para luego generar el modelo para la llamada de la API de CPLEX.

Para ambos modelos: se proponen las siguientes estructuras:

Tabla 4.1: Estructuras de datos para el modelo de caminos

Nombre	Tipo	Función principal
aristas	lista	almacenar todas las aristas del grafo dirigido
nodos	lista	almacenar todos los nodos
pesos	arreglo	almacenar todas las penalizaciones de cada nodo. *
costo	arreglo	almacenar todas las ponderaciones de cada arista. *

* en el caso de los arreglos, OPL no permite declarar arreglos dinámicamente, por lo que fue necesario generar una estructura intermedia en cada caso que almacene parcialmente los datos leídos para poder crear con ellas el arreglo posteriormente.

4.3. Formulación

Luego de tener generadas las estructuras, el proceso de transcribir el modelo a código es ágil cuando se tiene experiencia con OPLIDE. Es necesario definir las variables de decisión, la función objetivo y las propias restricciones, en un lenguaje muy similar a la notación matemática.

- La variable x_i que representa los nodos seleccionados es la más sencilla de las tres, es un arreglo de variables indexado por el identificador de nodo.

- La variable $x_{(i,j)}$ que representa las aristas seleccionadas, es un arreglo de variables indexado por el identificador de cada arista, esto es una tupla bi-dimensional.

- La variable $Y^{u,v}_{(i,j)}$ que representa las aristas seleccionadas de un camino entre u y v, es un arreglo de variables indexado por el identificador de cada camino, esto es una tupla tetra-dimensional.

Lamentablemente el modelo presenta una limitación referente a la demanda que acarrea generar las restricciones que tienen como actor la última variable. $Y^{u,v}_{(i,j)}$ depende cuadráticamente con la cantidad de nodos de la instancia, esto se verá en detalle en el Capítulo 5.

4.4. Datos de salida

Así como los datos de entrada están presentes en el formato STP, surgió la posibilidad de generar un formato para los archivos de salida que permita la visualización de las variables seleccionadas y las no seleccionadas, los caminos seleccionados entre todo par de nodos. Y además que brinde la posibilidad de exportar algunos datos sobre la ejecución, como el tiempo de cálculo entre otros.

El estándar está construido en particular para cubrir las necesidades de esta investigación, aunque es bastante flexible, debido a que respeta las pautas del formato JSON. Es posible acceder fácilmente a cada sección del archivo, comparar tiempos y como está pensado para trabajar utilizando el lenguaje JavaScript, lo hace ideal para poder visualizar los datos y dibujar el grafo en la web (Sección 4.5).

En la línea de comandos una opción interesante es utilizar el software JQ para acceder a las secciones de un archivo en este formato. El nombre optado para los archivos de salida es *PCM: Prize-Collecting Model*. En el apéndice 1 es posible ver detalles del estándar PCM.

4.5. Visualización

Este módulo era un plus necesario a la investigación, dado que en instancias grandes el simple hecho de controlar que la salida respetaba las condiciones de árbol implicaba tiempos desmedidos y extrema concentración. Con esta herramienta, el proceso de verificar que un grafo fuese árbol, pudo realizarse en muy poco tiempo incluso para grafos de más de 30 nodos.

Para la web se utilizó una biblioteca encargada de dibujar en un área de trabajo círculos y líneas, que en nuestro caso representan nodos y aristas, la biblioteca es Vis, que a su vez utiliza el framework AngularJs.

Para la interfaz se utilizó la biblioteca Materialize, que respeta el estándar Material Design para interfaces propuesto por Google.

Con las funcionalidades que presenta la web el usuario final puede: visualizar grafos (con los valores asociados a nodos y aristas), reordenarlos a voluntad y resaltar las aristas y nodos que fueron seleccionados. Además, en el caso de haber aplicado el modelo de caminos puede apreciarse cuál fue el camino seleccionado entre dos nodos y en el caso de haber aplicado el modelo de flujos, es posible visualizar los valores de flujo hasta llegar al nodo seleccionado. Otra funcionalidad interesante es poder comparar dos grafos: cual se resolvió más rápido, si presentan los mismos nodos y la cantidad de aristas.
El módulo fue bautizado como Prizewer: ***Priz***e Collecting Model Vi***ewer***.

4.6. Planteo final

Para ambos modelos, tanto el de caminos como el de flujos, la solución final consta de tres módulos:

1. El encargado de lectura de datos y generación de estructuras.
2. El encargado de crear el modelo, y su posterior resolución.
3. El encargado de exportar los datos al formato PCM.

Capítulo 5

Software a comparar

A continuación se detallan a fondo los aspectos clave de cada implementación. Posteriormente se describen los detalles más relevantes de los demás software a comparar en el Capítulo 6.

5.1. Implementación de ambos modelos

A continuación se realiza un análisis en profundidad de ambos modelos implementados. Para ambos modelos, dado un grafo de entrada $G(V, E, C, P)$ se denomina lo siguiente:

- $|V|$ = n la cantidad de nodos,
- $|E|$ = a la cantidad de aristas,
- $|P_+|$ = p la cantidad de nodos con penalización positiva P_+=$\{v \in V | p(v) > 0\}$.

5.1.1. Implementación: modelo de Flujos

- Existirán n valores para representar la variable y_i
- Existirán $2 * a + p$ valores para representar la variable $x_{(i,j)}$
- Existirán $2 * a + p$ valores para representar la variable $f_{(i,j)}$
- Existirá un vector de n entradas para representar la variable P que almacena las penalizaciones de cada nodo.
- Existirá un vector de $2 * a + p$ entradas para representar la variable C que almacena los costos de cada arista.

Con respecto a las restricciones del modelo, en base a las cardinalidades mencionadas se obtiene lo siguiente:

- Para la restricción (2.8), se generarán n instancias.
- Para la restricción (2.9), se generarán p instancias.
- Para la restricción (2.10), se generarán $n - p$ instancias.
- Para la restricción (2.11), se generarán $2 * a + p$ instancias.
- Para la restricción (2.12), se generará 1 instancia solamente.
- Para la restricción (2.13), se generarán $n+(2*a+p)+(2*a+p)$ instancias correspondiente a la suma del total de variables de decisión.

Sumado dan un total de: $(n) + (p) + (n - p) + (2 * a + p) + (1) + (n + (2 * a + p) + (2 * a + p))$ restricciones para cada instancia respectivamente.

5.1.2. Implementación: modelo de Camino

Análogo al caso anterior y en base a las cardinalidades mencionadas se obtiene lo siguiente:

- Existirán n valores para representar la variable xi_i
- Existirán $2 * a$ valores para representar la variable $X_{(i,j)}$
- Existirán $n * n - 1 * 2a$ valores para representar la variable $Y^{uv}_{(i,j)}$
- Existirá un vector de $2 * a$ entradas para representar la variable C que almacena los costos de cada arista.
- Existirá un vector de n entradas para representar la variable P que almacena las penalizaciones de cada nodo.

De igual forma y en base a las cardinalidades mencionadas se obtiene lo siguiente:

- Para la restricción (3.1), se generarán $n * (n - 1)$ instancias.
- Para la restricción (3.2), se generarán $n * (n - 1)$ instancias.
- Para la restricción (3.3), se generarán $n * (n - 1) * (n - 2)$ instancias.

- Para la restricción (3.4), se generarán $n * (n-1) * 2 * a$ instancias.
- Para la restricción (3.5), se generarán $2 * a$ instancias.
- Para la restricción (3.6), se generarán $2 * a$ instancias.
- Para la restricción (3.7), se generarán $n + 2a + n * (n-1) * 2a$ instancias.

Sumado dan un total de: $(n * (n-1)) + (n * (n-1)) + +(n * (n-1) * 2 * a) + (2 * a) + (2 * a) + (n + 2a + n * (n-1) * 2a)$ restricciones para cada instancia respectivamente.

5.2. Staynerd: mecanismo de resolución utilizando heurísticas

Otro gran aporte de la Dra. Ljubić a esta investigación, fue ceder una licencia para poder ejecutar el software staynerd descripto brevemente en el capítulo anterior.

Staynerd fue desarrollado en el seno de un equipo de investigadores especialistas en el área, enteramente dedicados a optimizar y volver sobre sus pasos buscando innovación y nuevos enfoques a la resolución de problemas.

El equipo encargado de su desarrollo, cuenta con varios de los integrantes que desarrollaron el software dhea (mencionado en el Capítulo 2), pero esta vez logrando un software más robusto, eficiente y general que su predecesor. El software utiliza un gran conjunto de diversas estrategias, activadas dependiendo de las características del grafo, esto lo hace sumamente eficiente a la hora de resolver instancias distintas. Los detalles referentes a la implementación están disponibles en Fischetti et al. (2016). Solamente resta comentar que antes de comenzar con la resolución de la instancia, se lleva a cabo un preprocesamiento de las características del grafo a resolver, y a partir de ellas es que se opta por resolver con un determinado procedimiento o con otro.

5.2.1. Tablas de clasificación y reglas

En las siguientes tablas se detallan las características y reglas de clasificación que utiliza staynerd para resolver las instancias.

En resumen estas tablas dan como resultado nada más ni nada menos que un árbol de toma de decisiones, que se utiliza como una hoja de ruta a la hora

Tabla 5.1: "Instance properties identified by the filter procedure" (Fischetti et al. (2016)). Aquí se listan las características a tener en cuenta en el proceso de clasificación

Property	Description
uniform	All arc weights have the same weight
sparse	Edge-to-arc ratio $\|E\|/\|V\| \leqslant 3$
dense	Edge-to-arc ratio $\|E\|/\|V\| \geqslant 3$
verydense	Edge-to-arc ratio $\|E\|/\|V\| \geqslant 5$
ratioT	Terminal ratio $\|T\|/\|V\|$
big	Number of nodes $\|V\| \geqslant 10,000$
small	Number of nodes $\|V\| \leqslant 1000$
hypercube	All nodes have the same degree
stp	Problem instance is of type STP
xy-model	The $(x, y) - model$ has been selected by the filter
bipartite	The instance graph G is bipartite w.r.t. the node sets V T and T

Tabla 5.2: "General filter rules" (Fischetti et al. (2016)). Aquí se listan los filtros con los que clasifica cada instancia a resolver

Rule	Applied settings
uniform $\wedge$ ($\neg$sparse $\vee$ ratioT $\leqslant 0.1$)	$\rightarrow y - model$
uniform $\wedge$ sparse $\wedge$ ratioT $\leqslant 0.1$	$\rightarrow (x, y) - model$
$\neg$uniform	$\rightarrow (x, y) - model$
dense	$\rightarrow$ Use tailing-off bound, high tolerance
verydense	$\rightarrow$ Use tailing-off bound, low tolerance

Tabla 5.3: "STP-specific rules" (Fischetti et al. (2016)). Uno de los desafíos era resolver instancias del STP básico

Rule	Applied settings
$xy - model \wedge$ ratioT $\leqslant 0.01$	$\rightarrow$ Separate dual ascent cuts
$xy - model \wedge$ ratioT $\leqslant 0.01$	$\rightarrow$ Add all dual ascent cuts
$xy - model \wedge$ ratioT $\leqslant 0.1 \wedge$ sparse $\wedge$ big	$\rightarrow$ Separate flow-balance constr., GSECs of size 2
$xy - model \wedge$ big $\wedge$ sparse	$\rightarrow$ Partition-based construction heuristic
verydense	$\rightarrow$ Preprocessing (special distance test)

Tabla 5.4: "Initialization heuristic rules" (Fischetti et al. (2016)). Aquí se listan las posibles heurísticas a aplicar

Rule	Applied settings
bipartite $\wedge$ uniform	$\rightarrow$ Set-covering heuristic
bipartite $\wedge$ $\neg$uniform $\wedge$ stp	$\rightarrow$ Blurred set-covering heuristic
hypercube $\wedge$ $\neg$uniform $\wedge$ $\neg$small $\wedge$ pcstp	$\rightarrow$ Blurred set-covering heuristic
$\neg$bipartite	$\rightarrow$ Local branching

de resolver determinadas instancias, y poder lograr mejores resultados, ambos aspectos tiempo y cercanía al óptimo. Tal como se comenta en su publicación, están basadas fuertemente en experiencia adquirida, generando un modelo de

clasificación a partir de un gran número de ejecuciones y características particulares de cada instancia.

5.3. Implementación factor 2 - Goemans and Williamson

Este algoritmo es el mismo descripto en la Sección 2.3. Fue implementado por Bauza and Olivera (2017) utilizando el lenguaje C++, en forma paralela a esta investigación. Para obtener más detalles sobre la implementación se sugiere leer su investigación.

Capítulo 6

Análisis Experimental

A continuación se detallan algunos de los aspectos a tener en cuenta referentes a las pruebas de rendimiento de las implementaciones. Posteriormente se realiza el estudio comparativo, aplicando diferentes criterios.

6.1. Entorno de pruebas

Las pruebas se realizaron en dos entornos distintos:

- **Entorno de Pruebas 1 (EP1)**: Intel core i5 460M 2.8 Ghz 3MB SmartCache, Corsair 8GiB DDR3 1066 MHz, NVIDIA GeForce 310M with 1GiB DDR3, Samsung 850 EVO 250 GiB SATA III.
- **Entorno de Pruebas 2 (EP2)**: Intel Pentium G3258 3.2 Ghz (O C 4.0Ghz) 3MiB SmartCache, GSKILL 16GiB DDR3 2133 MHz, NVIDIA GeForce 950 with 2GiB DDR5, Kingston 120GiB SSDNow V300 SATA III.

El desarrollo se llevó a cabo en paralelo en ambos entornos indistintamente, utilizando Git[1] como software para control de versiones.

6.2. Instancias DIMACS

En la página del desafío DIMACS se encuentra publicado un gran catálogo de instancias referentes al PCST. Comprenden instancias generadas de forma

[1] Git es un sistema de control de la versiones, destinado principalmente para el desarrollo de software y tareas afines

aleatoria, basadas en hipercubos, destinadas al análisis de redes biológicas, o simplemente escritas a mano. También es posible encontrar instancias de prueba para las demás variantes del problema de Steiner.

Para más información sobre las instancias se recomienda acceder a la sección descargas del sitio 11th DIMACS Implementation Challenge.

6.3. Scripts

Para poder realizar ejecuciones de forma masiva, fue necesario desarrollar scripts *bash*, que además de ahorrar mucho tiempo, brindaron la posibilidad de automatizar el proceso. Además se desarrollaron scripts que permitieron realizar la comparación de tiempos entre todos los ejecutables disponibles.

Como se mencionó anteriormente ya se contaba con un gran conjunto de instancias, pero fue necesario contar con algún mecanismo que permitiera generar instancias en formato STP, para poder realizar pruebas a medida. Con estos scripts se puede generar de forma aleatoria grafos que cumplan las condiciones necesarias para el PCST, de forma ágil. Las instancias generadas cumplen con las condiciones necesarias propuestas en la definición del problema: grafos no dirigidos, conexos, con ponderaciones asociadas a las aristas y penalizaciones asociadas a los nodos.

Por último, también fue necesaria otra variante que permitiera generar las instancias que se muestran en las Subsección 6.4.4 y Sección 6.5 con costos aleatorios.

6.4. Comparativa

Las comparaciones se dividirán en dos grandes grupos, para instancias pequeñas y posteriormente para instancias grandes. Esto se debe a que durante el proceso de desarrollo se notó que en algunos casos, la implementación del modelo de caminos no era capaz de generar el modelo necesario para poder realizar la llamada a la API de CPLEX (ver Sección 4.6). Esto se debe a la cantidad de restricciones generadas por el modelo, incluso a instancias relativamente pequeñas como se estudiará en la Subsección 6.4.1.

Consideraciones

A menos que se especifique lo contrario, el entorno de pruebas utilizado para la extracción de datos es EP2.

Los tiempos mencionados en todos los casos están representados en segundos y corresponde solamente a tiempos de cálculo sin contar el tiempo de preprocesamiento, generación del modelo y posterior generación de la salida.

6.4.1. EP1 vs EP2

Para comenzar la comparativa entre el modelo de flujos y el modelo de caminos, se realizó una prueba utilizando todas las instancias conexas en el rango de grafos conexos comprendido entre un K2 y un K11. Para cada una de estas instancias se realizaron 10000 iteraciones, en ambos entornos de prueba EP1 y EP2.

Como se puede apreciar en las tablas 6.1 y 6.2 el entorno EP2 fue siempre superior a EP1, dejando una clara evidencia de que la relación *tiempos - hardware*, como era de esperar.

Observaciones

Se propone la siguiente notación:

- Denotar como i a cada una de las instancias conexas comprendidas en el ranco K2-K11.
- Denotar como t_i al tiempo de cada iteración, correspondiente a la instancia i.
- Denotar como T_i a la suma de todos los t_i, correspondiente a la instancia i.

En las tablas 6.1 y 6.2 se muestra la suma de todos los T_i

En las figuras 6.1 y 6.2 se despliegan los valores de T_i.

Durante el proceso de análisis experimental, fue necesario encontrar un punto de referencia para las instancias donde entraba en conflicto la limitante presente en el modelo de caminos. De forma empírica, se realizaron pruebas al modelo ejecutando instancias cada vez más grandes.

Finalmente aplicando las ecuaciones enunciadas en la Subsección 5.1.2 se llegó a la siguiente conclusión:

- Para EP1: el tope se da entre las 51000 y 53000 restricciones, lo que corresponde a un 20x30[2], 19x33, 18x31, 17x40.

[2]Notación utilizada por el script, para definir el tamaño del grafo: NN**x**AA corresponde

Tabla 6.1: Comparativa modelo Caminos

Entorno	Tiempo total
EP1 → caminos	6917.284855
EP2 → caminos	3008.83504
Mejora	43.5 %

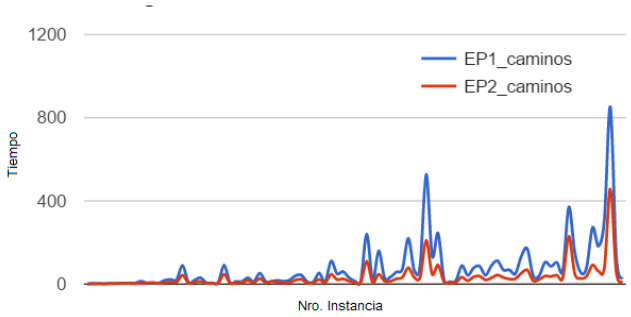
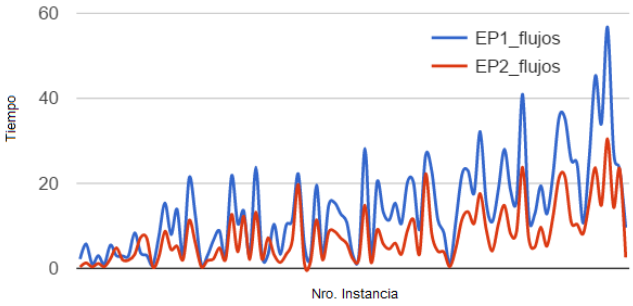


Tabla 6.2: Comparativa modelo flujos

Entorno	Tiempo total
EP1 → flujos	1285.06209
EP2 → flujos	704.102044
Mejora	54.79 %



- Para EP2: el tope se da entre las 74000 y 76000 restricciones, lo que corresponde a un 20x44[3], 19x50, 18x56, 17x62.

6.4.2. Instancias pequeñas hasta 20 nodos

Para estas pruebas se ejecutaron instancias generadas aleatoriamente. Dadas la cantidad de restricciones generadas por ambos modelos (estudiada en la Sección 5.1) y teniendo en cuenta que estas restricciones se reflejan en un mayor uso de memoria y procesador a la hora de ejecutar las pruebas, podemos plantear como hipótesis lo siguiente: *medida que las instancias sean más grandes, existirá una diferencia de tiempos cada vez mayor entre ambos modelos, manteniéndose siempre por debajo el modelo de flujos.*

a un grafo de NN nodos y AA aristas
[3]Ver nota anterior [2]

Camino vs Flujo

Para comenzar se ejecutaron pruebas con todas las instancias comprendidas entre un grafo K2 hasta llegar a un K9. Como es de esperar en todas las ejecuciones ambos modelos obtuvieron el mismo resultado y a su vez ha sido el óptimo. Sin embargo en los tiempos se aprecian algunas diferencias:

Tabla 6.3: Comparativa modelo flujos y caminos

Modelo	**Flujos**	**Caminos**	**Comp.***
Mejor ejecución (instancia)	0.0002368164 (n:5,a:8,p:1)	0.0007651367 (n:2,a:1,p:1)	30.95 %
Peor ejecución (instancia)	0.173659912 (n:8,a:16,p:7)	1.73349805 (n:9,a:35,p:8)	10.02 %
Promedio	0.01536426263	0.0661113722	23.24 %
Total (46000 ejecuciones)	706.7728118722	3041.1231229999	23.24 %

* Porcentaje de tiempo del modelo de flujos respecto del tiempo del modelo de caminos.

Dada la diferencia entre la cantidad de restricciones generadas por ambos modelos, a medida que aumenta la cantidad de nodos y aristas se observan mayores tiempos para el modelo de caminos a la hora de resolver el problema. Estos datos sirven para dar una primera aproximación a la comparativa, el mejor resultado en lo que a tiempos refiere para el modelo de caminos se dio en un grafo muy pequeño, respetando la hipótesis planteada al comienzo. Mientras que curiosamente la mejor ejecución de todas para el modelo de flujos fue un grafo intermedio, no cumpliéndose la relación antes mencionada restricciones-tiempo.

Para el peor caso, ocurre algo similar, es de esperar que en el modelo de caminos la ejecución con tiempo mayor sea producto de una instancia de las más grandes que se ejecutaron para estas pruebas. Para el caso del modelo de flujos, la instancia con mayor tiempo de ejecución es una instancia que tiene una alta relación en nodos con penalización positiva/nodos y como se mencionó anteriormente, esto afecta a este modelo incrementando la cantidad de restricciones.

El promedio, en cambio, es un estimativo más ajustado en este caso, de cuanto puede llegar a tardar una ejecución.

El total es para dar una idea de cuánto tiempo estuvo la CPU realizando cálculos para cada modelo. En total el modelo de flujos le tomó una fracción del 23.24 % del tiempo total que le tomó a su rival.

Si bien el modelo de caminos no acompaña la gran performance que tiene el modelo de flujos, al observar detalladamente estos tiempos se pueden ver ciertas particularidades como se apreciará en la Tabla 6.4:

Tabla 6.4: Instancias particulares: modelos caminos y flujos

Instancia	Parámetro	Caminos	Flujos
4x6	Mejor ejecución	0.0024821777	0.0024821777
	Peor ejecución	0.0417299805	0.0336621094
	Promedio	0.0096046768	0.0096638867
	Total (5000 ej)	4.8023383803	4.8319433604
6x10	Mejor ejecución	0.0153479004	0.0140898438
	Peor ejecución	0.0650029297	0.0627419434
	Promedio	0.0191818228	0.0174358355
	Total (5000 ej)	9.5909113776	8.7179177255
6x14	Mejor ejecución	0.0132109375	0.0172290039
	Peor ejecución	0.0591660156	0.084907959
	Promedio	0.0173507739	0.0225836079
	Total (5000 ej)	8.6753869636	11.2918039544
7x10	Mejor ejecución	0.0109609375	0.0210090332
	Peor ejecución	0.0415039062	0.0733391113
	Promedio	0.0136528379	0.0254064683
	Total (5000 ej)	6.8264189457	12.7032341317
7x21	Mejor ejecución	0.0331828613	0.0182578125
	Peor ejecución	0.0801691895	0.135977051
	Promedio	0.0463687021	0.0393614468
	Total (5000 ej)	23.1843510749	19.6807233892
9x27	Mejor ejecución	0.0445490723	0.0308190918
	Peor ejecución	0.175797119	0.162428955
	Promedio	0.059755043	0.0429583047
	Total (5000 ej)	29.8775214819	21.4791523441

Luego de estas ejecuciones, se realizaron una nueva serie de pruebas de hasta 5000 ejecuciones para descartar cualquier error o casualidad, obteniendo resultados muy similares a los obtenidos en esta ejecución. Una particularidad presente en estas instancias, está en los nodos con penalización positiva, presentan una elevada cantidad de enlaces hacia los demás nodos en relación a la cantidad de nodos totales del grafo. Esta particularidad, está condicionando a que se generen algunas restricciones adicionales en el modelo de flujos, que penaliza en parte su gran desempeño, llevando a perder la batalla con el modelo de caminos en algunos casos de forma abrumadora, como en el caso 7x10 donde además de salir victorioso en el 96 % de las ejecuciones. Al observar la suma total del tiempo necesario para terminar con todas las ejecuciones se aprecia que fue casi la mitad (6.8264189457 para el modelo de caminos y 12.7032341317 para el modelo de flujos).

Relajación a ambos modelos

Siguiendo la línea de la investigación presentada en Lucena and Resende (2004), en esta sección se realiza un análisis de relajación de ambos modelos para encontrar cotas inferiores a las instancias. Para la relajación se debe realizar un cambio en la lógica de cada respectivo modelo, para que las variables de decisión tomen valores continuos.

La ecuación (2.13), presentada en la Subsección 2.2.3, se modifica de la siguiente manera:

$$0 \leqslant X_{ij} \leqslant 1,\ 0 \leqslant y_i \leqslant 1 \quad \forall (i,j) \in A_{SA}, \forall i \in (V_{SA} \backslash \{r\})$$

La ecuación (3.7), presentada en la Sección 3.2, se modifica de la siguiente manera:

$$0 \leqslant X_{(i,j)} \leqslant 1,\ 0 \leqslant x_p \leqslant 1,\ 0 \leqslant Y_{(i,j)}^{u,v} \leqslant 1 \quad \forall p, u, v \in V_c, (i,j) \in E_c$$

En otras palabras; la relajación consiste en tomar las variables de decisión como continuas en el rango cerrado comprendido entre [0..1].

Una vez más, las instancias seleccionadas corresponden a grafos generados aleatoriamente desde dos y hasta de veinte nodos (limitando la ejecución en las 74000-76000 restricciones Subsección 6.4.1) para poder comparar ambos modelos.

Figura 6.1: Relajación: Gap costos

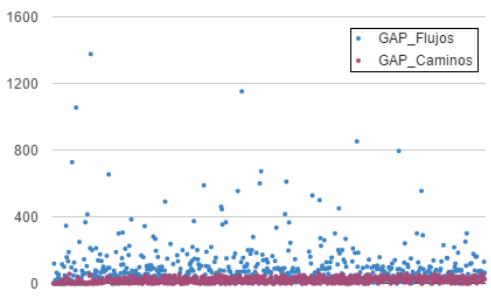
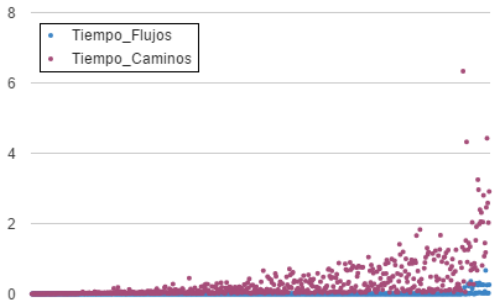


Figura 6.2: Relajación: Tiempos



En la Figura 6.1 se puede apreciar como el modelo de caminos presenta mejores cotas inferiores para la gran mayoría de las instancias, el promedio se puede visualizar en la Tabla 6.5

Respecto a la Figura 6.2 los tiempos de ejecución se puede notar que existe la misma curva presente para la solución con variables enteras [0,1].

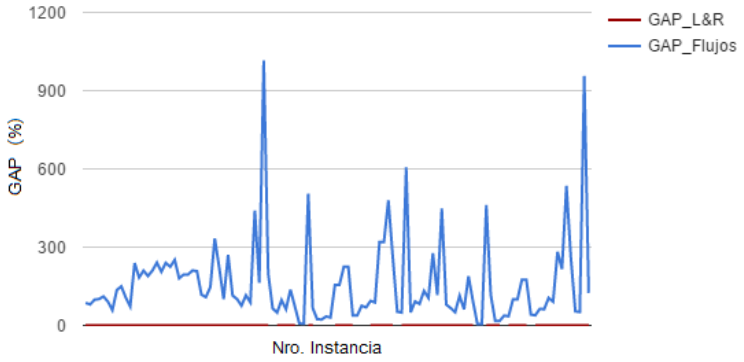
Tabla 6.5: Promedio de solución relajada

Modelo	Promedio
Modelo de caminos	91.15511463
Modelo de flujos	19.16939646

Por otro lado, también es posible comparar los resultados obtenidos al relajar el modelo de flujos con los datos presentes en Lucena and Resende (2004), implementación que se denotará como *L&R*.

Las instancias a ejecutar son algunas de las que fueron utilizadas en la competencia DIMACS.

Figura 6.3: Costos L&R y Flujos



En la Figura 6.3 se puede observar que el algoritmo utilizado en Lucena and Resende (2004) no culminó en algunas instancias, aunque en las instancias donde lo hizo, llegó a la solución óptima o en el peor de los casos tuvo un margen de diferencia de un 1.4 % respecto a la solución óptima. Mientras que el modelo de flujos, estuvo siempre por encima, alcanzando un gap de 1015 % en el peor caso.

No es posible comparar los tiempos, dado que las implementaciones se realizaron en distintos entornos y utilizando distintos lenguajes, siendo el único punto en común es que ambas utilizan CPLEX como herramienta de cálculo.

Aunque existe algo llamativo respecto a los tiempos de ejecución; el modelo de flujos nunca superó los 3 segundos, incluso para aquellas instancias donde la implementación L&R no finalizó. Y en el caso de la implementación L&R el promedio de ejecución fue de casi 8 horas teniendo en cuenta todas las instancias donde llegó a una solución, sobrepasando las 36 horas reiteradas ocasiones.

Algoritmos exactos vs algoritmos aproximados

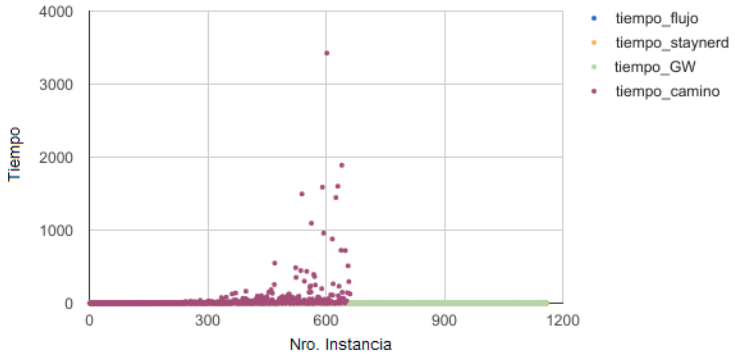
Esta sección se detallan los resultados de comparar ambos modelos implementados, en contraste con implementaciones cuya finalidad es encontrar soluciones aproximadas al problema a costo de ceder en el valor obtenido como solución, el cual puede no ser el óptimo.

Es necesario aclarar que en este apartado también se dividió la prueba en dos grandes grupos. Primero se ejecutaron instancias pequeñas, hasta las 74000 - 76000 restricciones para poder comparar los resultados con el modelo de caminos y posteriormente se realizaron ejecuciones más grandes, teniendo en cuenta el modelo de flujos como método exacto únicamente.

Análisis de tiempos - Instancias Pequeñas

En la Figura 6.4 es posible visualizar cómo el tiempo hace que sea prohibitivo utilizar el modelo de caminos para hallar una solución óptima a medida que crece el tamaño de la instancia.

Figura 6.4: Comparación todas las implementaciones juntas



Aunque para las primeras 150-200 instancias todas las implementaciones tuvieron un comportamiento similar como se puede apreciar en la Figura 6.5 este rango comprende desde un grafo K2 hasta un K11. Muy esporádicamente hay instancias del staynerd que sobresalen, pero lo más notorio es la curva que comienzan a tomar la disposición de los valores de la ejecución del modelo de caminos. Las demás implementaciones se mantienen cerca del 0.

En la Figura 6.6 se muestran casi los mismos datos que la figura anterior, con la diferencia de que se ocultan los valores de tiempo obtenidos en el modelo de caminos, reduciendo la escala lo que permite comparar las otras implementaciones.

Figura 6.5: Comparación todas las implementaciones hasta un K11

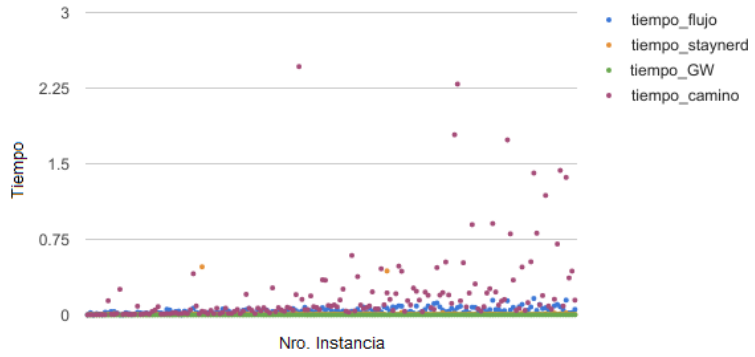
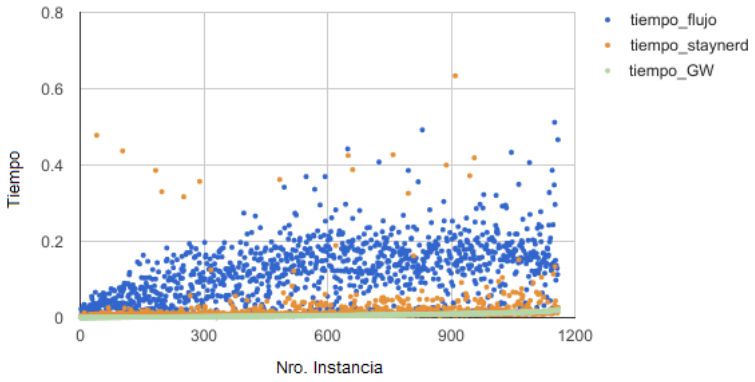


Figura 6.6: Comparación sin modelo de caminos



Se aprecia notoriamente que el modelo de flujos es quien tiene los tiempos de ejecución superiores, siendo superado muy esporádicamente por la implementación en base a heurísticas de staynerd, mientras que la implementación GW es la que se queda con los tiempos mínimos para todas las instancias.

Análisis de Costo - Instancias Pequeñas

La hipótesis para esta sección viene asociada a las implementaciones orientadas a encontrar la solución aproximada; deberían tener una *diferencia con respecto a la solución óptima*, dado que en el caso de GW garantiza tener un defasaje de hasta 2 veces respecto de la solución óptima. Y en el caso de staynerd, es de suponer que ocurrirá algo similar, al estar basada en heurísticas. También, es de esperar que ambos modelos presentados en esta investigación *lleguen al valor óptimo*.

En la Figura 6.7 se ven en distintos colores las soluciones óptimas de cada implementación, las que figuran con valor 0 respecto a la implementación de

Figura 6.7: Comparación flujos - staynerd

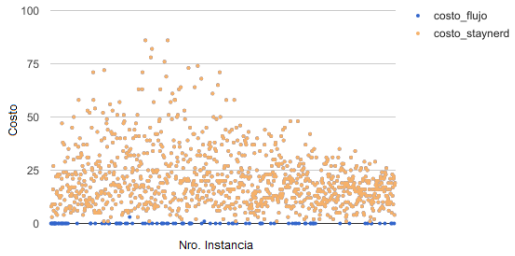
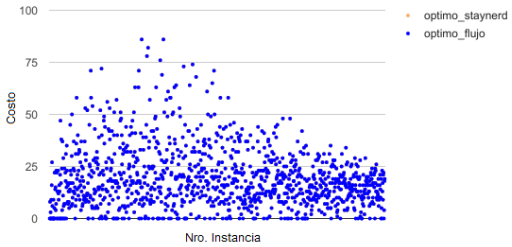


Figura 6.8: Comparación staynerd - flujos



flujos, corresponde a instancias (como ya se mencionó, generadas automáticamente) pero con una particularidad; son instancias donde se está ante un solo nodo con penalización positiva y aristas de salida con ponderación positiva, dejando como resultado la opción óptima de seleccionar este nodo y no seleccionar ninguna otra arista, obteniendo costo 0. Lo que sucede en estos casos dentro del código de la implementación staynerd es desconocido, pero no da salida alguna retornando el error *segmentation fault* y cortando la ejecución. Este caso es una situación muy particular a la cual no se hubiese llegado de no ser por haber desarrollado instancias propias para la implementación. Fuera de este inconveniente es posible comprobar mediante la Figura 6.8 que la implementación staynerd llegó al óptimo en todas las demás instancias (primero se dibujan los puntos de color amarillo y sobre ellos los puntos de color azul, no es posible ver ningún punto de color amarillo, todos se encuentran detrás de los azules).

En la Figura 6.9 se introducen los valores obtenidos para el modelo de caminos y se puede comprobar que efectivamente se obtuvo el óptimo en ambas soluciones, ya que coincide exactamente con los valores existentes para el staynerd y los valores existentes para el modelo de flujos, incluso los casos particulares descriptos en el párrafo anterior.

En la Figura 6.10, se agrega la implementación restante GW pudiéndose visualizar que en reiteradas oportunidades obtiene una diferencia respecto a la solución óptima. Esto sucede aproximadamente en 11.65 % de los las instancias, en promedio el gap entre la solución obtenida por su implementación y el óptimo ronda el 6,63 % manteniéndose siempre por debajo del factor 2 propuesto en el algoritmo general Goemans and Williamson (1995).

Figura 6.9: Comparación caminos - staynerd - flujos

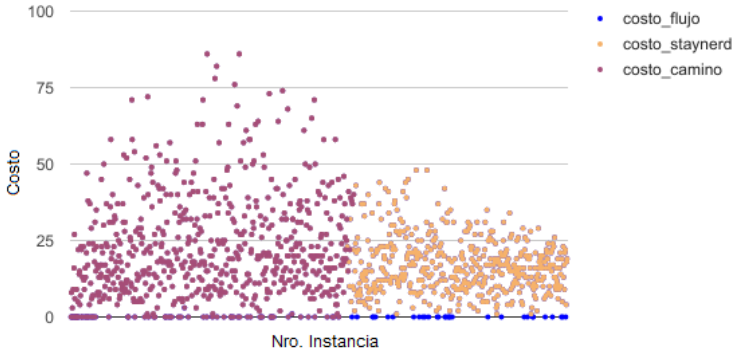
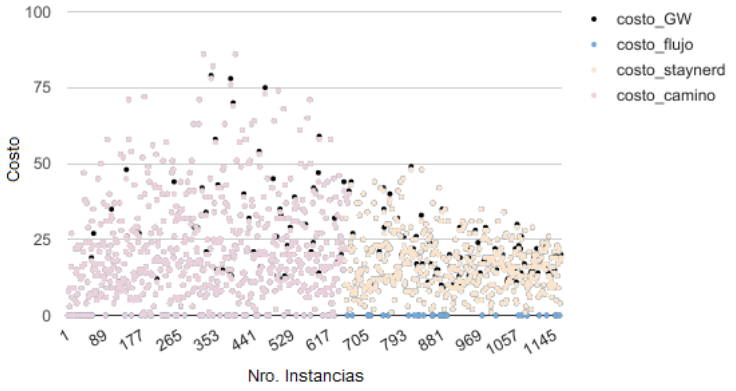


Figura 6.10: Comparación GW - flujos - staynerd - caminos



6.4.3. Instancias grandes

En esta sección se utilizaron las instancias de la competencia DIMACS mencionadas en la Sección 6.2, en este caso fueron sorteadas aleatoriamente resultando en 105 instancias diferentes, van desde grafos de 41 nodos y 625 aristas, hasta los 937 y 2000, pasando un variado espectro de densidades (ver definición 6.1). Para estas pruebas las implementaciones a tener en cuenta son tres: El modelo de flujos y ambas soluciones aproximadas GW y staynerd.

Un detalle a tener en cuenta; dado el tamaño de las instancias, se limitó el tiempo de todas las ejecuciones a una hora.

Análisis de tiempos - Instancias

El análisis de tiempo muestra claramente el costo extra que tiene realizar las pruebas con una solución de algoritmos exactos. En la Figura 6.11 se puede ver que el modelo de flujos terminó la ejecución por haber llegado al límite de tiempo en reiteradas ocasiones. Mientras que la implementación de algoritmos

de aproximación estuvo con un desempeño correcto y la implementación de heurísticas logró los mejores tiempos.

Figura 6.11: Comparación GW - flujos - staynerd

En la Tabla 6.6 se muestran algunos indicadores más detallados sobre la misma ejecución.

Tabla 6.6: Tiempos - Instancias grandes

Indicador	**Flujos**	**Staynerd**	**GW**
Promedio de ejecución	1932.213544321	2.2291938776	108.3231431566
Tiempo total de cálculo	204550.213199384	218.708	9107.330402

Para la siguiente prueba se utiliza el concepto de densidad.

Definición* 6.1** *Dado un grafo $G = (V, E)$, se define la* ***densidad de grafo *de la siguiente forma: $D = \frac{2|E|}{|V|\,(|V|-1)}$.*

Coleman and Moré (1983)

En base a la definición de *densidad de grafo* descripta en 6.1 al realizar el gráfico de tiempos respecto de su densidad, es posible apreciar que ninguna de las implementaciones tuvo variación dependiendo de la densidad del grafo, como se puede visualizar en la Figura 6.12.

Análisis de Costo

Con respecto a los costos, es posible visualizar en la Figura 6.13 (en este caso dado que el rango de valores obtenidos es muy extenso, se optó por mostrar el GAP de cada implementación), incluso con el límite de una hora el modelo de flujos es capaz de llegar al óptimo en reiteradas ocasiones. También se puede apreciar como el algoritmo de aproximación GW tiene fluctuaciones grandes en torno al óptimo. La implementación staynerd llegó al óptimo en todas las ejecuciones.

Figura 6.12: Comparación GW - flujos - staynerd

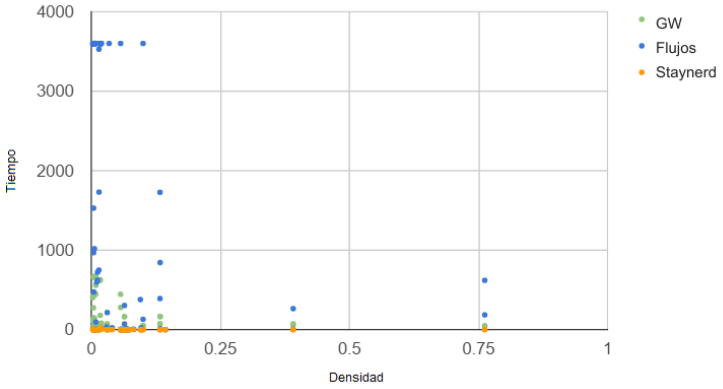


Figura 6.13: Comparación GW - flujos - staynerd

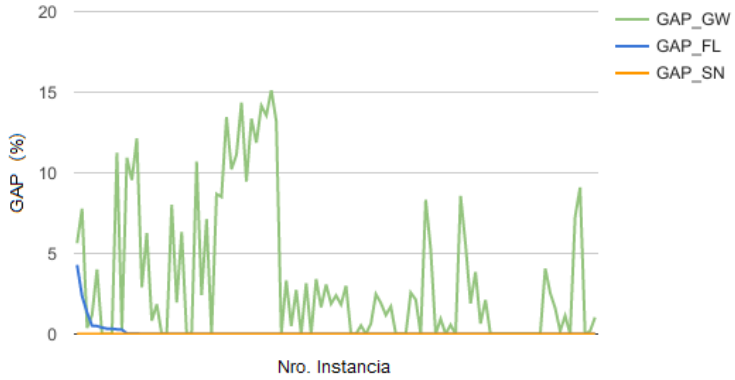


Tabla 6.7: Costos - Instancias grandes

Indicador	Flujos	Staynerd	GW
Promedio de GAP	0.102045333	0	3.557053139

En la Tabla 6.7 es posible apreciar como el staynerd logró alcanzar siempre el valor óptimo. En el caso del modelo de flujos, logró alcanzar la solución óptima en 13 de las 105 instancias y de esas 13 obtuvo 4.28 % de diferencia con el óptimo en el peor caso.

En la Tabla 6.8 se puede apreciar el tamaño de las instancias donde el modelo de flujos no logró alcanzar al óptimo. Es necesario remarcar que esto se debe a la limitante de tiempo impuesta al comienzo, la cual era de una hora. El valor devuelto en cada ejecución, corresponde a la mejor solución encontrada hasta ese momento. De ejecutar la misma instancia sin límite de tiempo, se hubiera encontrado la solución óptima.

Tabla 6.8: Instancias donde el modelo de flujos no llegó al óptimo

Nodos	Aristas	GW	Flujos	Staynerd
863	2000	71	70	67
512	2304	90	85	83
400	1442	476299	480932	474466
400	1426	394046	391481	389451
640	960	2740	2643	2630
400	1507	no_ejecutó	395794	394191
400	1470	no_ejecutó	492424	490771
400	1456	no_ejecutó	521255	519526
640	960	9167	8159	8135
400	1527	478238	478366	477073
640	960	47838	42620	42606

6.4.4. Topologías populares

Para estas pruebas, se seleccionaron varias topologías bien conocidas en la literatura Bertinat et al. (2015).

Plan de pruebas

Para estas topologías, se han seleccionado un juego de valores aleatorios para las aristas y nodos, siguiendo el esquema $(\alpha, 10\alpha)$ resultando 2 instancias para cada topología:

- **Aristas caras**: aristas con ponderación muy alta respecto al costo de los nodos
- **Aristas baratas**: aristas con ponderación muy inferior respecto al costo de los nodos

El plan de pruebas en este caso, es realizar las ejecuciones para ambas instancias, de todas las topologías seleccionadas y el resultado esperado dependerá de cada caso:

- **Aristas encarecidas:** se espera que sea seleccionado el nodo con mayor penalidad, ya que seleccionar una arista para llegar a otro nodo no es redituable, dado que la penalidad del nodo es menor al costo de la arista.
- **Aristas económicas:** se espera que sean seleccionados todos los nodos, ya que seleccionar una aristas para llegar al siguiente nodo es más económico respecto a la penalidad que se debe pagar por no seleccionarlo.

RAU2 - Aristas encarecidas

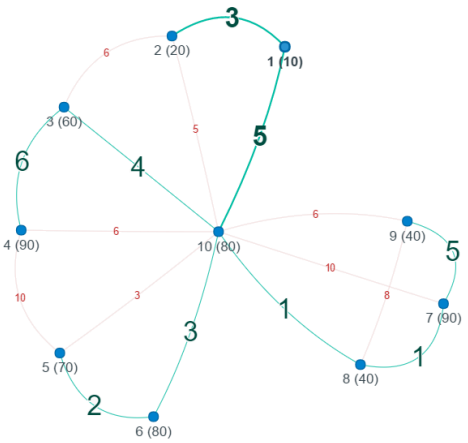


Figura 6.14: Modelo caminos

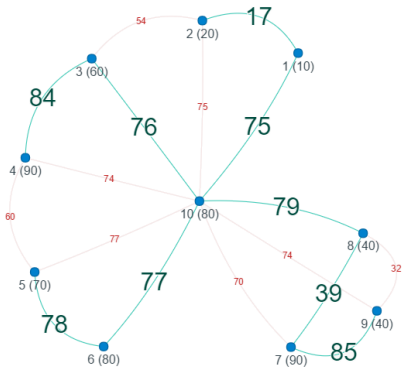


Figura 6.15: Modelo flujos

En este caso, el grafo presenta similares características y además se cumplió la hipótesis, al seleccionarse todas las aristas. La solución óptima en ambos casos tiene un valor de 30.

RAU2 - Aristas económicas

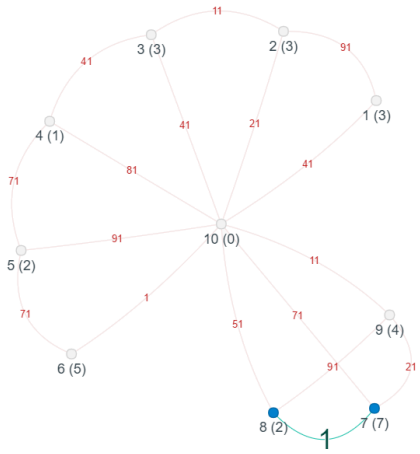


Figura 6.16: Modelo caminos

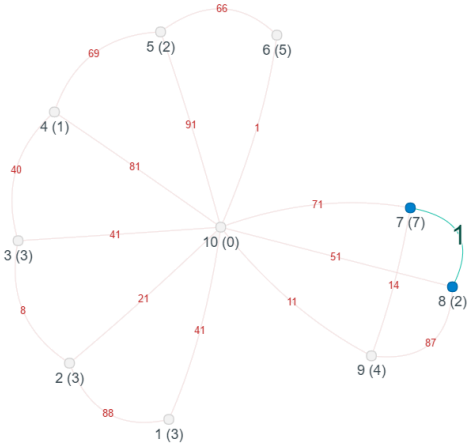


Figura 6.17: Modelo flujos

En este caso sucedió algo curioso, el sorteo para la ponderación de las aristas y las penalidades de los nodos dio como resultado que la solución óptima sea un subgrafo de dos nodos y una arista, debido a una arista con ponderación menor a las penalidades de sus extremos.

Arpanet - Aristas encarecidas

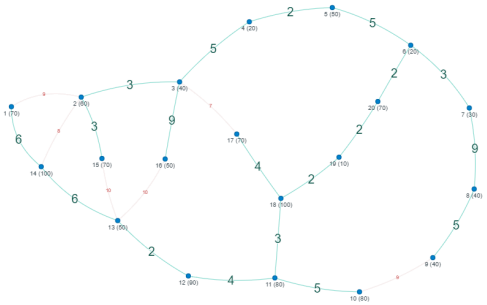


Figura 6.18: Modelo caminos

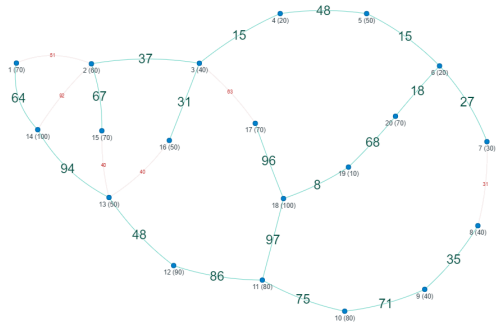


Figura 6.19: Modelo flujos

Similar al anterior caso, ambos modelos dieron como resultado el mismo grado, con costo 80.

Aristas económicas

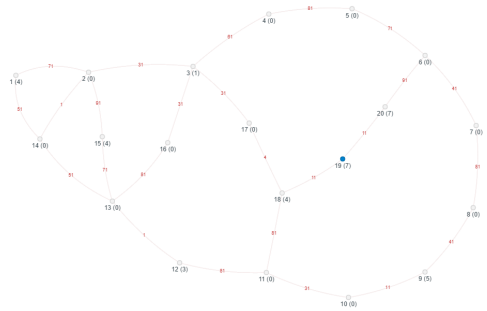


Figura 6.20: Modelo caminos

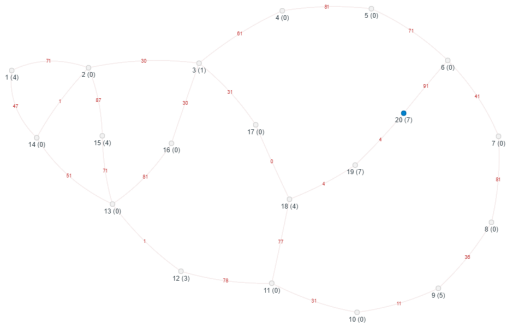


Figura 6.21: Modelo flujos

La solución óptima únicamente presenta un único nodo, el de mayor penalidad con un valor de 7, aunque existían dos nodos con esa penalidad y los modelos los seleccionaron intercalados.

Dodecahedro - Aristas encarecidas

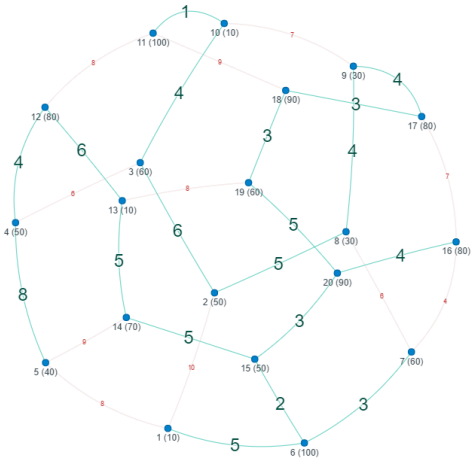


Figura 6.22: Modelo caminos

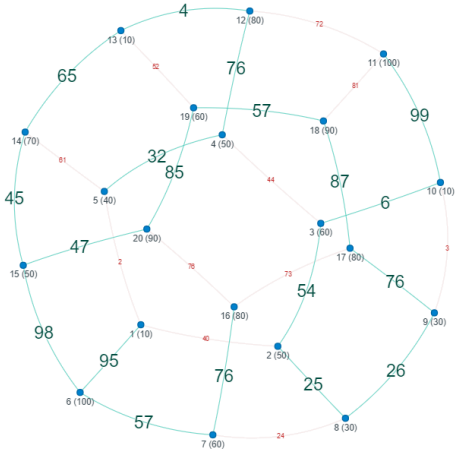


Figura 6.23: Modelo flujos

En este caso si bien ambos modelos llegaron al mismo costo (80), los subárboles generados no son iguales, dado que el árbol generado por el modelo de caminos tiene tres vértices con grado 3 y en el caso del árbol generado por el modelo de flujos tiene solamente dos vértices con esas características.

Dodecahedro - Aristas económicas

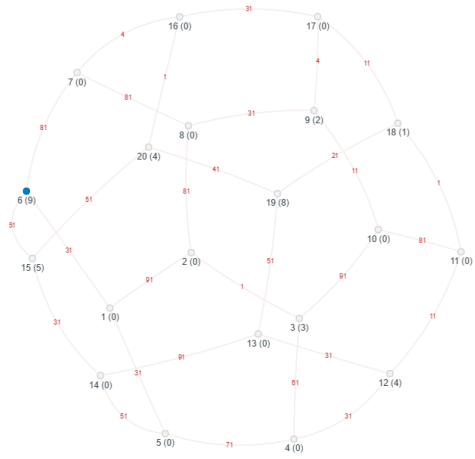


Figura 6.24: Modelo caminos

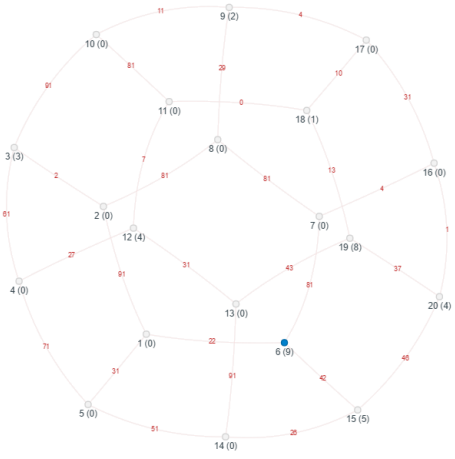


Figura 6.25: Modelo flujos

En este caso, vuelve a suceder lo esperado, solo es seleccionado un nodo.

6.4.5. Aplicación a redes de ANTEL

En estos casos, solamente es aplicable el modelo de flujos, dado que ambas instancias generan más restricciones que las que son capaces de ejecutar en EP2 con el modelo de caminos.

Antel red oeste

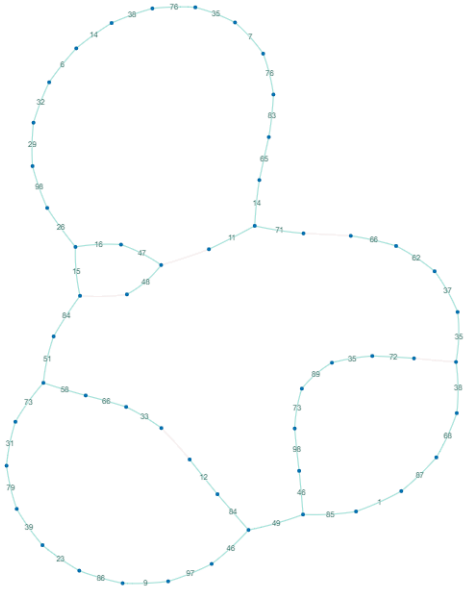


Figura 6.26: Aristas económicas

Figura 6.27: Aristas encarecidas

En esta topología, en la Figura 6.26 nuevamente se aprecia como fueron seleccionados todos los nodos, el costo total de esta ejecución fue de 241.

En la Figura 6.27 se observa como fue seleccionado un nodo solamente, dejando un costo de 30, producto de pagar la penalización de todos los nodos no seleccionados.

Antel red este

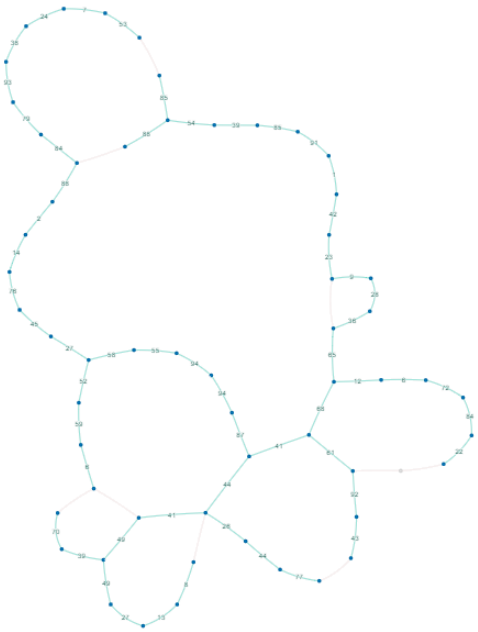


Figura 6.28: Aristas económicas

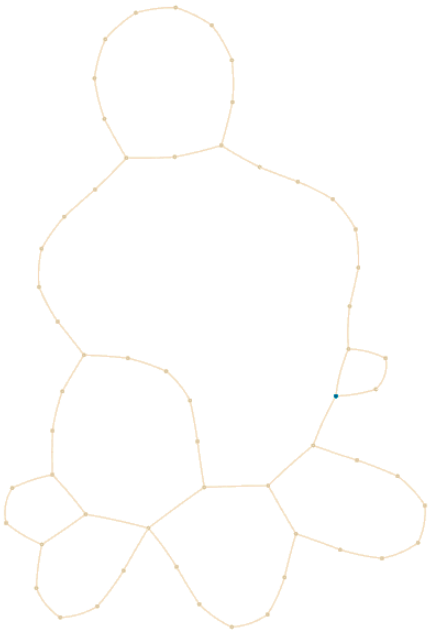


Figura 6.29: Aristas encarecidas

En estos casos se presentan las mismas situaciones de que la topología de la Sección 6.4.5 Antel oeste. Dejando los costos 281 para la Figura 6.28 y 25 para la Figura 6.29

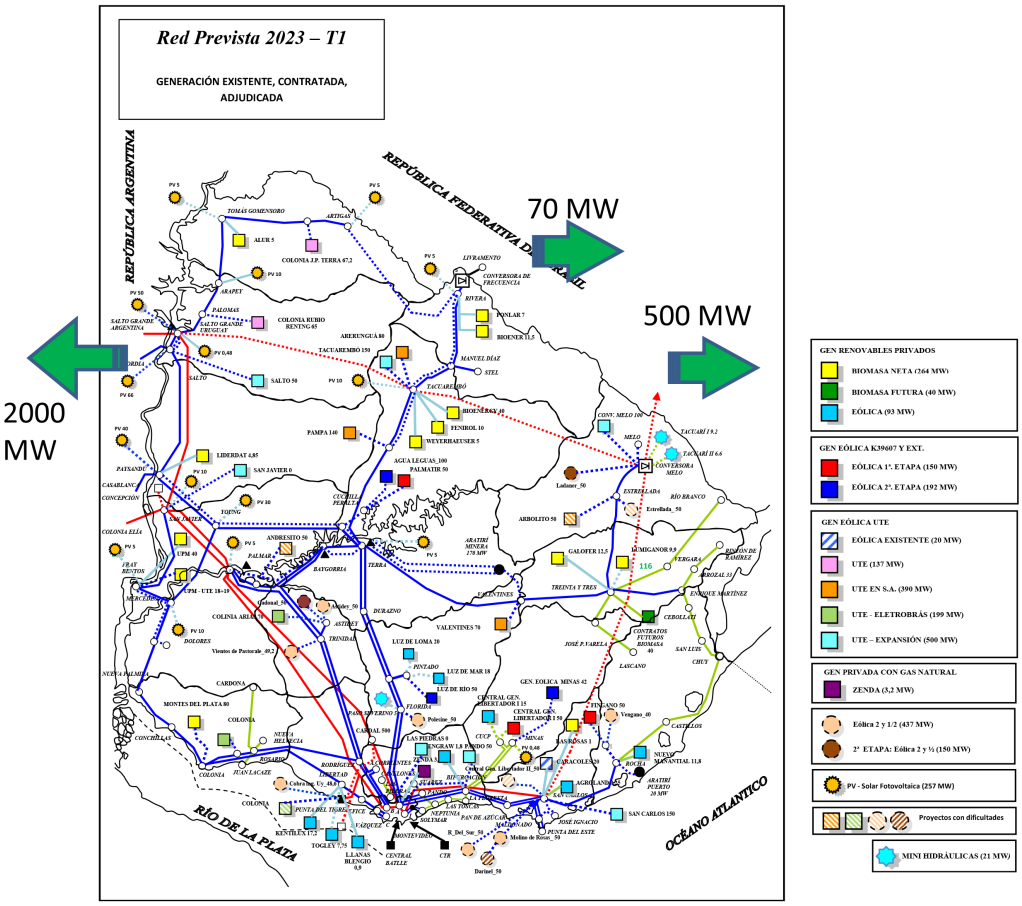
6.5. Aplicación a UTE

En esta sección se tomará como caso de estudio la red eólica planificada por UTE para el año 2023. Los datos fueron extraídos de la sección *UTE i*[4].

6.5.1. Instancias

En la Figura 6.30[5] se muestra la red general esperada para el 2023 que incluye parques públicos y privados entre ellos además de los parques eólicos se aprecian plantas de biomasa, generación por gas natural, parques fotovoltaicos, y plantas de generación hidráulica.

Figura 6.30: UTE: red prevista para 2023

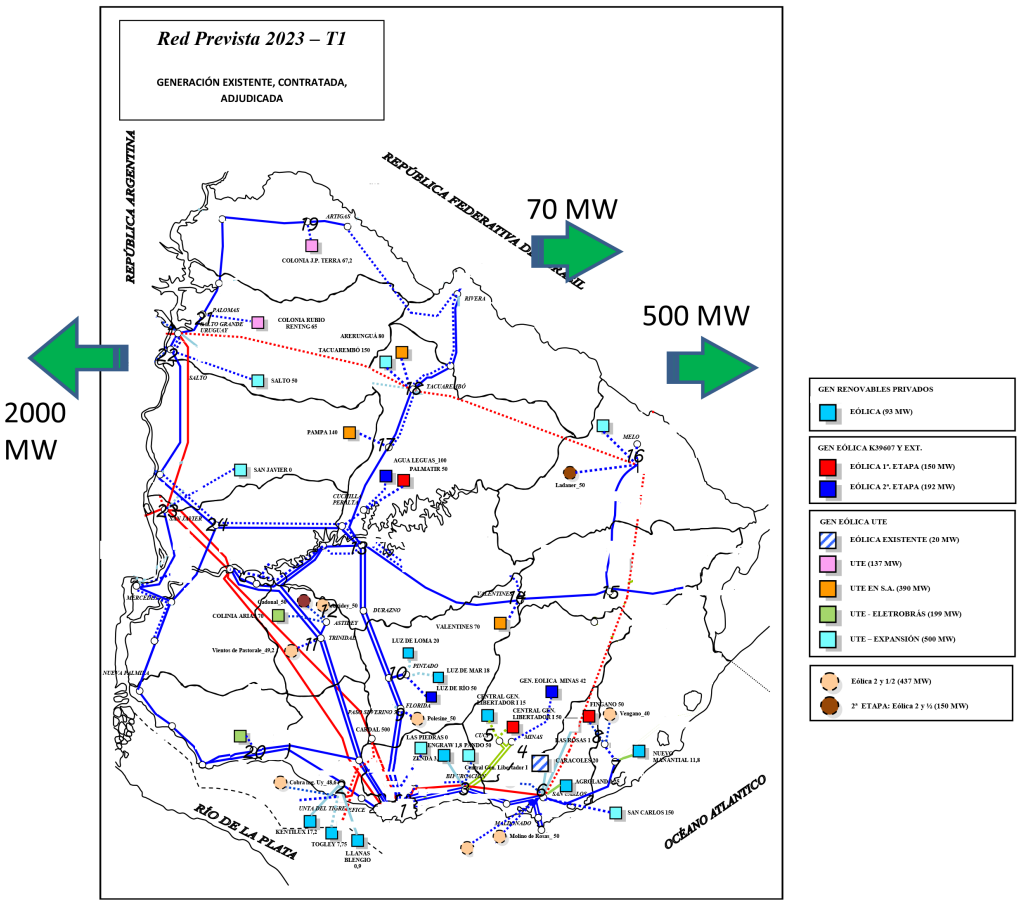


[4]UTE i es sinónimo de INFORMACIÓN, INNOVACIÓN, INVERSIÓN, INTELIGENCIA, INCLUSIÓN, INVESTIGACIÓN, INFRAESTRUCTURA e INTERACCIÓN. (http://portal.ute.com.uy)

[5]http://portal.ute.com.uy/sites/default/files/documents/files/mapa%202023.pdf

Una versión simplificada de la red anterior se presenta en Figura 6.31.

Figura 6.31: UTE: solo red eólica



En la Figura 6.32 se muestra la red eólica agrupada por sectores de generación, dejando tres nodos en los que no están representados parques de generación eólica, aunque representan puntos estratégicos en la red: Treinta y Tres, Young y Montevideo.

Dado que en la red original los parques eólicos ya están dispuestos de acuerdo a un plan estratégico para 2023, resulta interesante estudiar la disposición aleatoria de 4 nuevas conexiones entre los parques de la red, Figura 6.33.

Figura 6.32: Red eólica (2023) simplificada

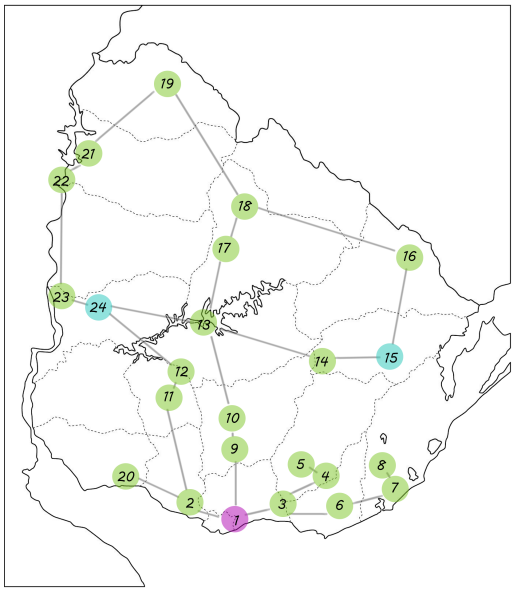
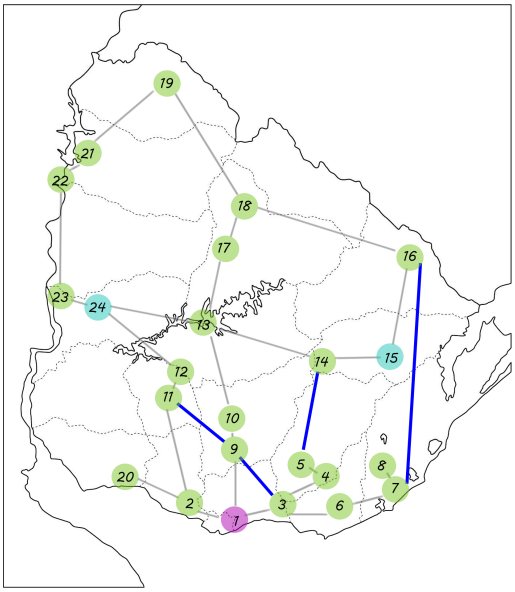


Figura 6.33: Red eólica (2023) con aristas aleatorias



Observaciones:

- Las penalizaciones asignadas a los vértices del grafo corresponde a la capacidad de generación sumada de los parques que componen la agrupación.
- En todos los casos se asignó valores aleatorios en un rango acotado $[1, 2, .., min]$, siendo min la menor penalización de todas las asignada a los vértices.

El objetivo de sortear valores aleatorios es para comprobar que el modelo funciona y puede ser aplicabdo en esta topología independientemente de los valores asignados para esta prueba.

6.5.2. Ejecuciones

Tabla 6.9: Ponderación aristas

Arista	**Pond.**
(19,21)	100
(18,19)	120
(20,2)	90
(21,22)	100
(22,23)	70
(23,24)	50
(15,16)	105
(16,18)	115
(17,18)	60
(13,17)	95
(13,14)	80
(14,15)	85
(1,2)	50
(1,3)	60
(1,9)	65
(2,11)	120
(3,4)	76
(3,6)	54
(4,5)	8
(6,7)	25
(7,8)	45
(9,10)	63
(10,13)	71
(11,12)	33
(12,24)	130
(24,13)	96
(11,9)*	77
(14,5)*	77
(7,16)*	146
(3,9)*	117

El símbolo * denota las aristas introducidas aleatoriamente.

Tabla 6.10: Penalización nodos

Nodo	**Penaliz.**
1	0
2	716
3	593
4	592
5	243
6	1487
7	93
8	587
9	437
10	378
11	437
12	786
13	242
14	390
15	0
16	650
17	390
18	890
19	137
20	199
21	137
22	500
23	500
24	0

Las figuras 6.34 y 6.35, muestran el resultado de la ejecución con los datos de la red planificada para 2023 (Ver tablas 6.10 y 6.9). En ambos casos las ejecuciones dieron el mismo resultado, 1500.

Figura 6.34: Modelo de caminos, red 2023

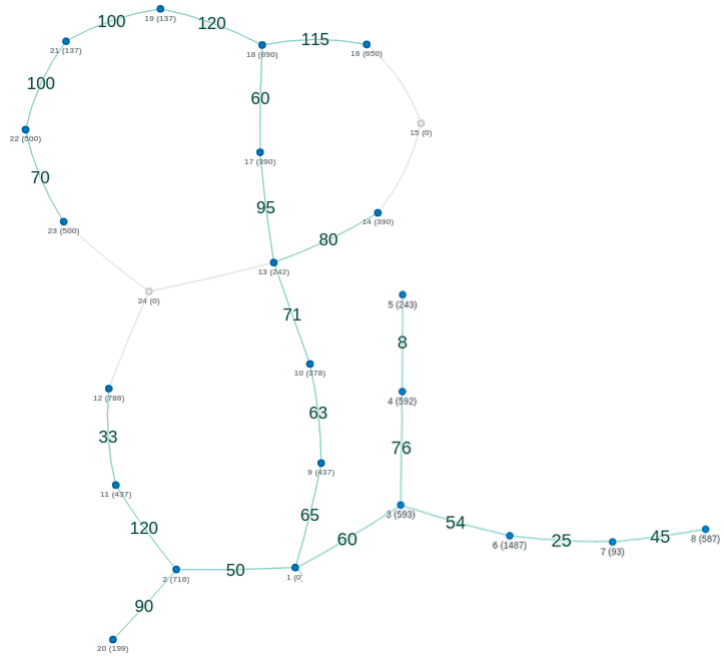
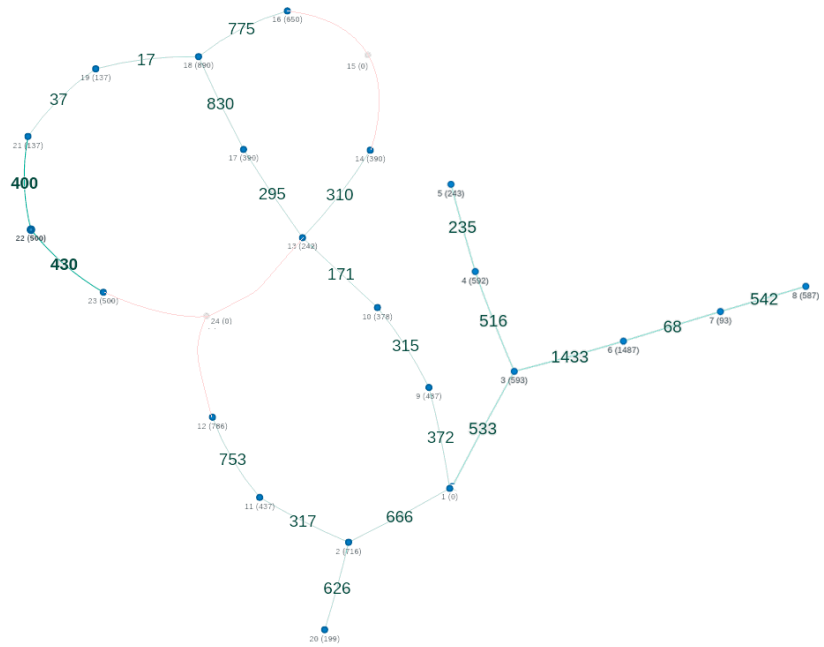


Figura 6.35: Modelo de flujos, red 2023



Las figuras 6.36 y 6.37, muestran el resultado de la ejecución con los datos de la red planificada para 2013. En ambos casos las ejecuciones dieron el mismo resultado, 1454. El resultado es un poco menor, producto de haber seleccionado las aristas generadas aleatoriamente.

Figura 6.36: Modelo de caminos, red 2023 + 4 aristas aleatorias

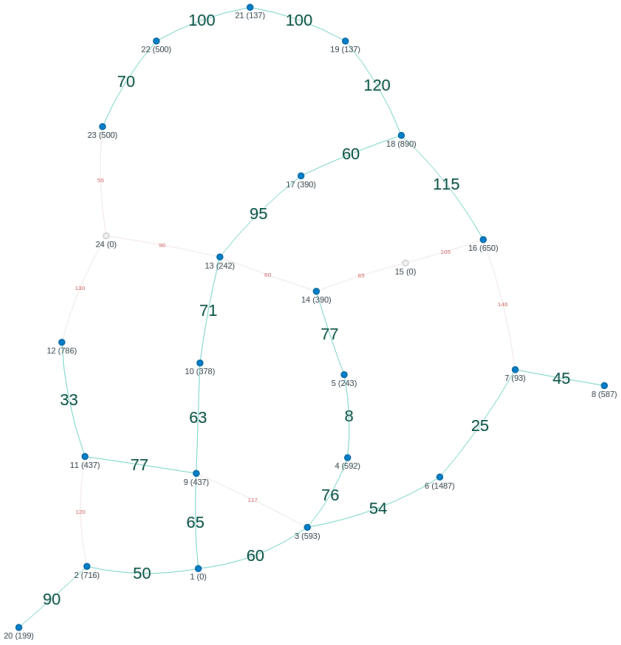
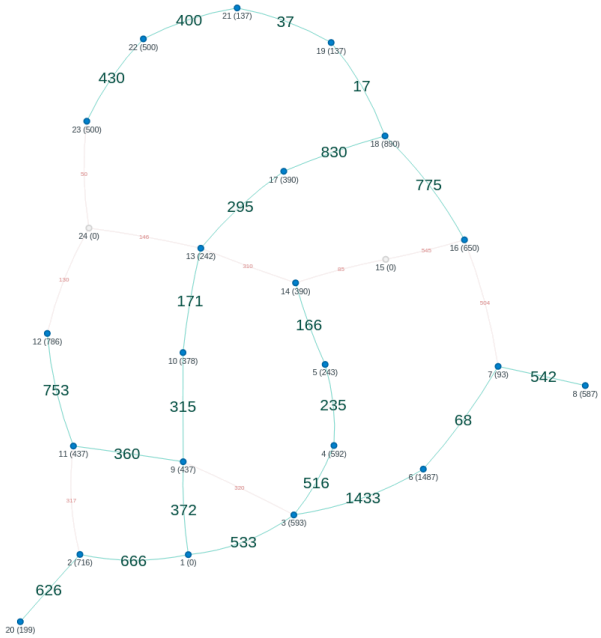


Figura 6.37: Modelo de flujos, red 2023 + 4 aristas aleatorias



6.6. Resultados

En esta sección se comentan los resultados obtenidos a partir de las ejecuciones, se contraponen tiempos así como las variaciones en los costos y algunas comentarios del problema.

Primero comentar como un gran punto a favor, el haber logrado que ambas implementaciones cumplen con lo propuesto al comienzo y llegan al óptimo, en tiempos razonables en muchos casos, teniendo en cuenta sus limitaciones. Ambos modelos presentan diferencias importantes entre las cardinalidades totales a la hora de generar las restricciones. Una consecuencia directa de ello es provocar tiempos prohibitivos aún para instancias pequeñas, con fluctuaciones de hasta una hora para el modelo de caminos, e incluso no ejecutándose en muchos casos debido a que no pudo generarse el modelo de ejecución necesario para la llamada a CPLEX.

Ambas implementaciones se desempeñaron mejor a medida que el entorno de pruebas tenía más recursos, lo que hace suponer que el modelo de caminos puede llegar a resolver alguna de las instancias propuestas en el DIMACS implementation challenge. Esto sería interesante investigar dado que al relajar ambos modelos se obtuvieron mejores cotas utilizando el modelo de caminos que el modelo de flujos.

A la hora de comparar con un software de referencia como es el staynerd uno de los ganadores del desafío DIMACS en el área de heurísticas, la implementación de flujos se comporta correctamente, incluso siendo una implementación de un algoritmo exacto. Los tiempos del staynerd son generalmente menores, aunque esporádicamente no se respetaba esta lógica, de hecho quien obtuvo el peor tiempo de ejecución fue staynerd con un total de 0.634 segundos para un grafo 19x94 en comparación con el peor caso del modelo de flujos con un tiempo de 0,512047119 segundos para un grafo 20x182. Además es importante comentar que staynerd no es capaz de llegar a la solución en reiteradas oportunidades, dando errores de ejecución.

Respecto a la implementación GW, en este caso se pudo comprobar lo que se mencionaba al comienzo de la investigación, es necesario tener presente que es lo que se está buscando a la hora de resolver el problema. Si interesa obtener una solución en un tiempo menor y se está dispuesto a ceder en el valor obtenido, la solución viene por el lado de una implementación que utilice algoritmos aproximados y en particular se mostró que el algoritmo de Goemans

and Williamson es una opción interesante. Pero si lo que se busca es encontrar la mejor solución al problema, se debe optar por un algoritmo exacto. Esta necesidad es probable que suceda en situaciones reales donde cada decisión a tomar refleja: costos importantes, depende el futuro de una empresa, o una inversión abultada pero con alto riesgo, lo primordial es tomar la mejor opción y en estos casos puede ser conveniente tomarse un tiempo extra para resolver el problema, obteniendo una solución más acertada.

Para instancias un poco más grandes (hasta 625 nodos y 2000 aristas), es posible notar que la solución exacta sigue siendo una opción interesante incluso con límite de tiempo. Al mirar las tablas 6.7 y 6.6 puede verse que en algunas ocasiones para tiempos no muy superiores al obtenido con la solución aproximada, el modelo de flujos se comporta sorprendentemente llegando cerca del 87,62 % de las ejecuciones al valor óptimo, lo que da como resultado un promedio del 0.1 %.

Respecto a las instancias particulares mencionadas en Bertinat et al. (2015) es posible comprobar que ambos modelos tienen un funcionamiento bien distinto, se cumplieron las hipótesis esperadas, más allá de alguna eventualidad (RAU2 - Aristas encarecidas, Figura 6.16) surgida por haber generado aleatoriamente la instancia.

En particular, para caso de la red de parques eólicos planteada para 2013, mediante la prueba realizada en la Subsección 6.5.2 se comprobó que el modelo funciona y puede ser aplicable en esta topología independientemente de los valores asignados para esta prueba.

6.7. Problemas

La principal dificultad de esta investigación, yace en la implementación de caminos, lamentablemente la implementación tiene la limitante de tener muy poco margen para las restricciones, cuyo número va de la mano con los recursos que tenga el dispositivo que está ejecutando la instancia. De esto es posible extraer como positivo el gran aprendizaje que se logró de la herramienta CPLEX, aspecto que luego sirvió durante proceso de desarrollo, para obtener un panorama más amplio del background que está presente en esta herramienta y no es visible hacia el usuario final.

Por otro lado, durante el proceso de implementación surgió un oportunidad (mediante una promoción provista por IBM) de utilizar el entorno de desarrollo

en la nube de IBM, la plataforma lleva el nombre de *DOcplexcloud*[6]*: Plataforma destinada a la ejecución de modelos Implementados utilizando CPLEX.* Se ofrece la posibilidad de ejecutar remotamente el modelo en sus servidores. Aunque al comienzo fue una gran idea, existía una limitante; esta plataforma solo permite ejecutar modelos donde únicamente se ingresen datos utilizando un formato propio de OPL (archivos DAT) o Planillas de *Excel*. Para poder utilizar los archivos STP que contenían las instancias para ejecutar tanto en el modelo de caminos, como en el de flujos, era necesario poder subir uno de estos archivos, o cualquier archivo de texto plano para poder procesarlo posteriormente. Hubiese sido de gran ayuda poder contar con esta opción, ya que la plataforma DOcplexcloud brinda entornos especialmente diseñados para ejecutar modelos implementados utilizando CPLEX.

Otro aspecto desafortunado; IBM tiene un plan para estudiantes, el cuál tiene como finalidad brindarle acceso a estudiantes para que hagan uso de su herramienta, sin embargo el proceso resulta ser muy enrevesado. Hubiese sido de gran ayuda contar con el software en etapas tempranas de la investigación, lo que quizá podría haber sido de gran ayuda para identificar problemas como el que se listó al comienzo.

[6] DOcplexcloud

Capítulo 7

Consideraciones finales

7.1. Conclusiones

El PCST es un problema que resume en gran medida diversas realidades en algo tan simple como un grafo ponderado. No es difícil deducir por qué es el elegido cuando se trata de optimización en redes eléctricas, telecomunicaciones y otros. Es entonces donde surge la necesidad de resolver el problema, y atacarlo utilizando estrategias que garanticen la solución óptima, suele ser lo primero a considerar. Es allí donde entran en juego los algoritmos exactos, siendo el único mecanismo garantiza la llegada al óptimo.

Los problemas *NP-Difícil* tienen una complejidad muy elevada, desde el momento de comprender el problema, hasta contar con un método para resolverlo, como es el caso del PCST.

Existe una variada cantidad de investigadores que han propuesto sus ideas y con ellas sus modelos de resolución.

En esta investigación, se ha resuelto el problema desarrollando un modelo de programación lineal entera y resolviéndolo mediante una de las herramientas más eficientes en el área de optimización como lo es CPLEX.

En este caso el aspecto interdisciplinario jugó un rol importante; pudiéndose aplicar una estrategia bien conocida en el área de circuitos eléctricos, llevándolo a lo que se introdujo como índice de Kirchhoff.

El comportamiento no fue muy performante, pero se pudo demostrar que en todas las instancias donde se logró generar el modelo, la implementación cumplió su cometido de resolver el problema hallando la solución óptima.

Con respecto a la comparativa con otras implementaciones del estado del

arte, se mostró que ambas implementaciones tuvieron una performance correcta, teniendo en cuenta que estaban frente a muy buenos rivales. Es bueno destacar en este punto el aspecto de haber contactado con la Dra. Ivana Ljuvić que brindó material muy interesante, así como información sobre la competencia DIMACS.

Se logró aplicar el modelo a la red de parques eólicos planificada del territorio uruguayo, así como otras topologías bien conocidas, demostrando la versatilidad del nuevo modelo introducido en esta investigación.

Además mencionar la utilidad de la herramienta Prizewer, un punto que no había sido planeado al comienzo, pero a medida que se avanzaba con la implementación y las ejecuciones se hacía cada vez más necesaria.

En resumen, se cumplió con los objetivos planteados al comienzo; comprender el problema, estudiar las soluciones planteadas por investigadores de muy prestigiosas universidades, desarrollar un modelo capaz de resolverlo, verificar que efectivamente se cumplía el objetivo de llegar al óptimo y comparar su desempeño con varias implementaciones. Es muy satisfactorio hacer una revisión de todo lo logrado y poder afirmar que durante todo el transcurso de esta investigación ha sido un proceso de aprendizaje constante.

7.2. Trabajo futuro

Existen algunos aspectos en los que sería muy interesante investigar:

Probar los modelos implementados en otros ambientes: En la competencia DIMACS, todas las implementaciones fueron ejecutadas en servidores con características muy superiores a las utilizadas en esta investigación. Si bien fue una ventaja contar con una de las implementaciones más galardonadas, es probable que los modelos se comporten de otra forma, y se obtengan mejores resultados. La primer opción es el servicio provisto por IBM, por lo que sería necesario desarrollar algún mecanismo que traduzca las instancias en formato STP al formato DAT aceptado por la plataforma.

Mejorar lo actual: En particular respecto al modelo de caminos; implementar una rutina de preprocesamiento como la mencionada en Ljubić et al. (2004) que permita identificar nodos que no estarán en la solución antes de generar el modelo, reduciendo la cantidad de restricciones necesarias para resolverlo.

Otro modelo: En caso que no sea posible lo anterior, sería buena idea

utilizar la experiencia adquirida durante la investigación, para crear un nuevo modelo, que permita realizar una comparación de tiempos con instancias aún más grandes.

PCM: Promover el uso del estándar aceptado por Prizewer para la visualización de nodos así como agregar más funcionalidades, para obtener un producto que pueda utilizarse para más problemas.

Cálculos en GPU: Aprovechar los grandes avances que han tenido los procesadores de las GPU en los últimos años, transformándolos en verdaderos procesadores para cómputo masivamente paralelos. En el foro oficial de CPLEX, se ha mencionado que no se prevé presentar esta funcionalidad a futuro y en consecuencia habría que implementar el modelo de forma tal que pueda ser ejecutado utilizando esta tecnología.

7.3. Recomendaciones

En esta sección se detallan, desde un humilde punto de vista, algunos consejos y recomendaciones para las personas que piensen investigar este problema o similares lo siguiente:

- Para comenzar, el PCST es un problema ampliamente estudiado y como se presentó en la Sección 2.1, ha sido atacado con distintas técnicas, las cuales a su vez se han mejorado con el paso del tiempo. En las secciones 6.4.4, 6.4.5 y 6.5 se mostró que este problema es muy versátil, lo que efectivamente comprueba lo planteado al comienzo de esta investigación Sección 1.1. Estos aspectos, hacen que el PCST sea una opción muy interesante y difícil de descartar en ámbitos donde estén presentes las características principales del problema.
- A a la hora de trabajar con alguna de las variantes del problema de Steiner (y en algunos casos más generales donde sea necesario trabajar con grafos) se use el formato STP, el cual es muy versátil, muy utilizado y muy bien documentado.
- Para resolver problemas de optimización, utilizar la herramienta CPLEX ahorra mucho tiempo de pruebas. IBM cuenta con muchos científicos, ingenieros y personas afines en constante avance para aplicar las técnicas más avanzadas para la resolución de problemas. También es recomendable utilizar en conjunto con el entorno de desarrollo OPLIDE. Ambos presentan muy buena documentación y cuentan comunidad de desarrollo

muy activa.

- Como último punto, el código del Prizewer se hará publico luego de la defensa de este proyecto, puede ser interesante utilizar alguna de las bibliotecas que se utilizaron para su desarrollo.

Referencias bibliográficas

Akhmedov, M., Kwee, I., and Montemanni, R. (2016). A divide and conquer matheuristic algorithm for the prize-collecting steiner tree problem. Computers & Operations Research, 70:18 – 25.

Archer, A., Bateni, M., Hajiaghayi, M., and Karloff, H. (2011). Improved approximation algorithms for prize-collecting steiner tree and tsp. SIAM Journal on Computing, 40(2):309–332.

Backes, C., Rurainski, A., Klau, G., Müller, O., Stöckel, D., Gerasch, A., Küntzer, J., Maisel, D., Ludwig, N., Hein, M., Keller, A., Burtscher, H., Kaufmann, M., Meese, E., and Lenhof, H.-P. (2012). An integer linear programming approach for finding deregulated subgraphs in regulatory networks. Nucleic Acids Research, 40(6):1–13.

Balas, E. (1989). The prize collecting traveling salesman problem. Networks, 19(6):621–636.

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. Fundamenta Mathematicae, 3(1):133–181.

Bauza, C. and Olivera, P. (2017). Tesis de grado, Ingeniería en computación, Modelos y Herramientas para Problemas de Planificación bajo Incertidumbre. Repositorio Colibrí, Udelar.

Beasley, J. E. (1984). An algorithm for the steiner problem in graphs. Networks, 14(1):147–159.

Bertinat, M. E., Cancela, H., González, M. F., Robledo, F., and Romero, P. (2015). Statistical methods for diameter constrained reliability estimation

in rare event scenarios. In 2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM), pages 243–250.

Bertsimas, D. and Sim, M. (2004). The price of robustness. Operations Research, 52(1):35–53.

Biazzo, I., Braunstein, A., and Zecchina, R. (2013). On the performance of a cavity method based algorithm for the prize-collecting steiner tree problem on graphs. CoRR, abs/1309.0346.

Bienstock, D., Goemans, M. X., Simchi-Levi, D., and Williamson, D. (1993). A note on the prize collecting traveling salesman problem. Mathematical Programming, 59(1):413–420.

Bonchev, D., Balaban, A. T., Liu, X., and Klein, D. J. (1994). Molecular cyclicity and centricity of polycyclic graphs. i. cyclicity based on resistance distances or reciprocal distances. International Journal of Quantum Chemistry, 50(1):1–20.

Brazil, M., Graham, R. L., Thomas, D. A., and Zachariasen, M. (2014). On the History of the Euclidean Steiner Tree Problem. Archive for History of Exact Sciences, 68:327–354.

Canuto, S. A., Resende, M. G. C., and Ribeiro, C. C. (2001). Local search with perturbations for the prize-collecting steiner tree problem in graphs. Networks, 38(1):50–58.

Chapovska, O. and Punnen, A. P. (2006). Variations of the prize-collecting steiner tree problem. Networks, 47(4):199–205.

Clayton, P. (2001). Fundamentals of Electric Circuit Analysis. John Wiley & Sons.

Coleman, T. F. and Moré, J. J. (1983). Estimation of sparse jacobian matrices and graph coloring blems. SIAM Journal on Numerical Analysis, 20(1):187–209.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, pages 151–158, New York, NY, USA. ACM.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). Introduction to Algorithms. McGraw-Hill Higher Education, 2nd edition.

Courant, R. and Robbins, H. (1941). What is Mathematics?: An Elementary Approach to Ideas and Methods. Oxford University Press, London.

da Cunha, A. S., Lucena, A., Maculan, N., and Resende, M. G. (2009). A relax-and-cut algorithm for the prize-collecting steiner problem in graphs. Discrete Applied Mathematics, 157(6):1198 – 1217. Reformulation Techniques and Mathematical Programming.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America, 2(4):393–410.

Dantzig, G. B. (1947). Maximization of a linear function of variables subject to linear inequalities. T.C. Koopmans, ed., Wiley, New York, pages 339–347.

Dittrich, M. T., Klau, G. W., Rosenwald, A., Dandekar, T., and Müller, T. (2008). Identifying functional modules in protein–protein interaction networks: an integrated exact approach. Bioinformatics, 24(13):i223.

Feofiloff, P., Fernandes, C. G., Ferreira, C. E., and de Pina, J. C. (2007). Primal-dual approximation algorithms for the prize-collecting steiner tree problem. Information Processing Letters, 103(5):195–202.

Fischetti, M., Leitner, M., Ljubić, I., Luipersbeck, M., Monaci, M., Resch, M., Salvagnin, D., and Sinnl, M. (2016). Thinning out steiner trees: a node-based model for uniform edge costs. Mathematical Programming Computation, pages 1–27.

Goemans, M. X. and Williamson, D. P. (1995). A general approximation technique for constrained forest problems. SIAM J. Comput., 24(2):296–317.

Gouveia, L. and Pesneau, P. (2006). On extended formulations for the precedence constrained asymmetric traveling salesman problem. Netw., 48(2):77–89.

Gouveia, L. and Pires, J. M. (1999). The asymmetric travelling salesman problem and a reformulation of the miller–tucker–zemlin constraints. European Journal of Operational Research, 112(1):134 – 146.

Gouveia, L. and Pires, J. M. (2001). The asymmetric travelling salesman problem: on generalizations of disaggregated miller–tucker–zemlin constraints. Discrete Applied Mathematics, 112(1–3):129 – 145. Combinatorial Optimization Symposium, Selected Papers.

Haouari, M., Layeb, S. B., and Sherali, H. D. (2008). The prize collecting steiner tree problem: models and lagrangian dual optimization approaches. Computational Optimization and Applications, 40(1):13–39.

Haouari, M., Layeb, S. B., and Sherali, H. D. (2010). Algorithmic expedients for the prize collecting steiner tree problem. Discrete Optimization, 7(1–2):32 – 47.

Haouari, M., Layeb, S. B., and Sherali, H. D. (2013). Tight compact models and comparative analysis for the prize collecting steiner tree problem. Discrete Applied Mathematics, 161(4–5):618 – 632. Seventh International Conference on Graphs and Optimization 2010.

Haouari, M. and Siala, J. C. (2006). A hybrid lagrangian genetic algorithm for the prize collecting steiner tree problem. Computers & Operations Research, 33(5):1274 – 1288.

Johnson, D. S., Minkoff, M., and Phillips, S. (2000). The prize collecting steiner tree problem: Theory and practice. In Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '00, pages 760–769, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Karp, R. M. (1972). Reducibility among Combinatorial Problems, pages 85–103. Springer US, Boston, MA.

Klau, G. W., Ljubić, I., Mutzel, P., Pferschy, U., and Weiskircher, R. (2003). The Fractional Prize-Collecting Steiner Tree Problem on Trees, pages 691–702. Springer Berlin Heidelberg, Berlin, Heidelberg.

Koch, T., Martin, A., and Voß, S. (2001). SteinLib: An Updated Library on Steiner Tree Problems in Graphs, pages 285–325. Springer US, Boston, MA.

Könemann, J., Sadeghabad, S. S., and Sanità, L. (2013). An LMP o(log n)-approximation algorithm for node weighted prize collecting steiner tree. CoRR, abs/1302.2127.

Krarup, J. and Vajada, S. (1997). On torricelli's geometrical solution to a problem of fermat. IMA Journal of Management Mathematics, 8(3):215–224.

Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G., Mutzel, P., and Fischetti, M. (2004). Solving the prize-collecting steiner tree problem to optimality. Technische Universitat Wien, Institut für Computergraphik und Algorithmen.

Ljubić, I., Weiskircher, R., Pferschy, U., Klau, G. W., Mutzel, P., and Fischetti, M. (2006). An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. Mathematical Programming, 105(2):427–449.

Lucena, A. (2005). Non delayed relax-and-cut algorithms. Annals of Operations Research, 140(1):375–410.

Lucena, A. and Resende, M. (2004). Strong lower bounds for the prize collecting steiner problem in graphs. Discrete Applied Mathematics, 141(1–3):277 – 294. Brazilian Symposium on Graphs, Algorithms and Combinatorics.

Mézard, M. and Parisi, G. (2003). The cavity method at zero temperature. Journal of Statistical Physics, 111(1):1–34.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. J. ACM, 7(4):326–329.

Nash, J. F. (1950). Non-coperative games. PhD thesis, Princeton.

Neumann, J. V. and Morgenstern, O. (1944). Theory of Games and Economic Behavior. princeton.

Parodi, C. (2011). Magister en ingeniería matemática integer optimization applied to the design of robust minimum cost multi-layer networks. Master's thesis, Facultad de Ingeniería, Universidad de la República.

Pedemonte, M. (2009). Tesis de maestría en informática ant colony optimization para la resolución del problema de steiner. Master's thesis, Facultad de Ingeniería, Universidad de la República.

Romero, P. (2009). Monografía, licenciado mención matemática, diámetro confiabilidad de una red. Repositorio Colibrí, Udelar.

Salamon, P., Sibani, P., and Frost, R. (2002). Facts, Conjectures, and Improvements for Simulated Annealing. Society for Industrial and Applied Mathematics.

Sarin, S. C., Sherali, H. D., and Bhootra, A. (2005). New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. Operations Research Letters, 33(1):62 – 70.

Schrijver, A. (2002). On the history of the transportation and maximum flow problems.

Segev, A. (1987). The node-weighted steiner tree problem. Networks, 17(1):1–17.

Sherali, H. D. and Driscoll, P. J. (2002). On tightening the relaxations of miller-tucker-zemlin formulations for asymmetric traveling salesman problems. Oper. Res., 50(4):656–669.

Sherali, H. D., Sarin, S. C., and Tsai, P.-F. (2006). A class of lifted path and flow-based formulations for the asymmetric traveling salesman problem with and without precedence constraints. Discrete Optimization, 3(1):20 – 32. The Traveling Salesman Problem.

Talbi, E. G. (2009). Metaheuristics from design to implementation, volume 1. Wiley Publishing.

Tuncbag, N., McCallum, S., Huang, S.-s. C., and Fraenkel, E. (2012). Steinernet: a web server for integrating 'omic' data to discover hidden components of response pathways. Nucleic Acids Research, 40(W1):W505.

Vardi, I. (1991). Computational recreations in Mathematica. Addison-Wesley, Redwood City, Calif. [u.a.].

Vazirani, V. V. (2001). Approximation Algorithms. Springer-Verlag New York, Inc., New York, NY, USA.

Williamson, D. P. and Shmoys, D. B. (2011). The Design of Approximation Algorithms. Cambridge University Press, New York, NY, USA, 1st edition.

Xiao, W. and Gutman, I. (2003). Resistance distance and laplacian spectrum. Theoretical Chemistry Accounts, 110(4):284–289.

Álvarez Miranda, E., Ljubić, I., and Toth, P. (2013). Exact approaches for solving robust prize-collecting steiner tree problems. European Journal of Operational Research, 229(3):599 – 612.

APÉNDICES

Apéndice 1

Formatos Utilizados

En esta sección se describe el formato de datos (tanto de entrada como de salida) utilizado para las instancias.

1.1. Formato STP

Algunos comentarios sobre el formato:

- El formato es orientado a filas.
- El fin de cada línea es determinado por un fin de línea.
- Los comentarios se realizan con un símbolo de #.
- Las líneas en blanco se ignoran.
- Para garantizar que un archivo `.stp` hace referencia a un archivo válido para ejecutarse como entrada en alguna de las variantes del problema de Steiner, la primer línea del archivo debe ser: `"33D32945 STP File, STP Format Version 1.0"`

El archivo se divide en secciones. Cada sección comienza con la palabra clave `SECTION` seguido del nombre de la sección, posteriormente el contenido y termina con una línea que contiene la palabra clave `END`.

El archivo finaliza con la palabra clave `EOF`.

Las posibles secciones se muestran en la Tabla 1.1 y tienen que aparecer en el orden dado.

Al ser esta una investigación referente al PCST, de todas las entradas disponibles solamente utilizan las siguientes:

Tabla 1.1: Secciones de un archivo .STP

Sección	Finalidad
`Comment`	Proporciona información general sobre la instancia del problema, como su nombre y su creador.
`Graph`	Aquí se especifica el grafo en sí, la cantidad de nodos y aristas, las cuales también se detallan en esta sección)
`Terminals`	Enumera los terminales de la instancia de problema.
`Coordinates`	Esta es una sección necesaria únicamente en casos donde se quiere representar gráficamente.

Tabla 1.2: Secciones de un archivo .STP

Entrada	Parámetros	Finalidad
`Nodes`	1	Detallar el número total de nodos del grafo.
`Edges`	1	Detallar el número total de Aristas del grafo
`E`	3	Declarar la arista del grafo, con su respectiva ponderación
`TP`	2	Declarar que un determinado nodo, posee penalización positiva

Se pueden consultar los detalles en Koch et al. (2001)

1.1.1. Ejemplo

Ejemplo de datos de entrada presentados en el formato STP

```
33D32945 STP File, STP Format Version 1.0

SECTION Comment
Name    "instanciaRandom"
Creator "generaGrafo"
Problem "Prize-Collecting Steiner Problem in Graphs"
END

SECTION Graph
Nodes 2
Edges 1
E 1 2 9
END

SECTION Terminals
Terminals 1
TP 2 99
END

EOF
```

1.2. Formato PCM

Algunos comentarios sobre el formato:

- Planteado en base al estudio de el Prize-Collecting Steiner Tree.
- Utiliza el formato JSON, por lo que se puede leer como texto siemple.
- Los comentarios se pueden realizar de la forma usual a como funcionan en el formato JSON.
- Sirve para visualizar datos propios de la ejecución como datos del grafo solución.

Tabla 1.3: Secciones de un archivo .PCM

Atributo	**Tipo**	**Finalidad**
`Tipo`	`ENUM`	Detallar si corresponde a un resultado obtenido por el modelo de caminos o de flujos.
`Costo`	`INT`	Detallar el costo obtenido en esa ejecución.
`Relajada`	`BOOL`	Detallar si la ejecución corresponde a variables de decisión relajadas o no.
`nodos`	`OBJECT`	Detallar respecto a la totalidad de nodos; su identificador, su penalización, y si fue seleccionado o no
`aristas`	`OBJECT`	Detallar respecto a la totalidad de aristas; sus extremos, su ponderación, si fue seleccionada, y por último si es solución de flujos; aparece una variable extra que indica el flujo total que pasa por esa arista.
`camino`	`OBJECT`	Detallar la totalidad de caminos, únicamente válida si el tipo es caminos

1.2.1. Ejemplo

Ejemplo de datos de salida producto de una ejecución (del modelo de caminos en este caso) en el formato PCM.


```
{"instancia":"gG_2x1.stp",
"tiempo":"0.00548999023",
"tipo":"camino",
"costo":0,
"relajada":false,
"xi":
[{"id":1,"prize":9,"selected":1}
,{"id":2,"prize":0,"selected":0}
],
"xij":
[{"from":2,"to":1,"cost":6,"selected":0}
],
"yuijv":
[]
}
```


* Notar que se encuentra presente un campo `"yuijv"` para desplegar los caminos seleccionados

Ejemplo de datos de salida producto de una ejecución (del modelo de flujos en este caso) en el formato PCM.


```
{"instancia":"gG_2x1.stp",
"tiempo":"0.0015769043",
"tipo":"flujo",
"costo":0,
"relajada":false,
"xi":
[{"id":1,"prize":9,"selected":1}
,{"id":2,"prize":0,"selected":0}
],
"xij":
[{"from":2,"to":1,"cost":3,"selected":0,"flow":0}
]}
```


*Notar que ya no aparece el campo `"yuijv"`, pero si aparece un campo extra `"flow"` en la sección aristas

Apéndice 2

Prizewer!

2.1. Visualización

La visualización se realiza mediante un navegador, Prizewer está desarrollado principalmente con contenido estático, por lo que se puede correr localmente, por esta razón no necesita de un servidor, aunque si se necesitara por algún motivo puede desplegarse sin mayores problemas utilizando cualquiera de ellos, como por ejemplo PHP, nodeJS, o python.

2.1.1. Capturas de pantalla

Figura 2.1: Página principal

Figura 2.2: Visualización

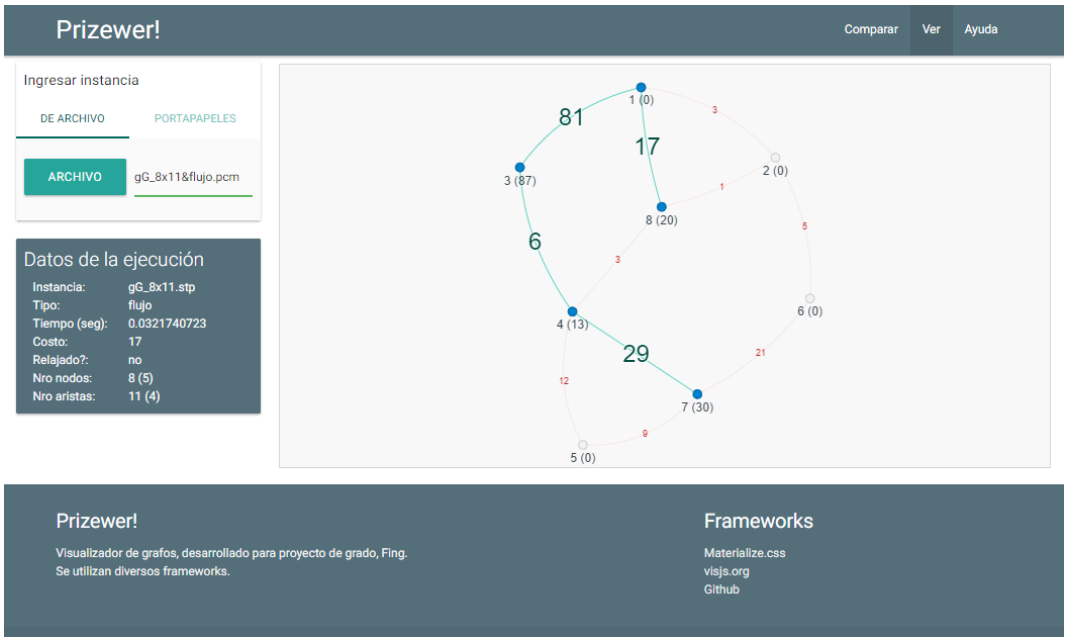


Figura 2.3: Comparación

Figura 2.4: Ayuda y más información