UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Diseño de Redes 2-Nodo-Conexas de Mínimo Costo con Nodos de Steiner y Caminos Acotados (2NCON-BP-SN)

Martín Piperno

28 de junio de 2017

Facultad de Ingeniería
Universidad de la República
Montevideo - Uruguay.

Agradecimientos

Este proyecto está dedicado a mis tutores por todo el apoyo durante todo el transcurso del mismo. Así como también a mi familia y amigos que me apoyaron durante toda la carrera.

Por otra parte, también dedico este documento a la memoria de mi padre, el cual de alguna forma siempre estará conmigo.

Resumen

El objeto de estudio es un problema de optimización de aplicabilidad en telecomunicaciones. Se dispone de una red con costos en sus aristas, la cual cuenta con nodos de Steiner opcionales con costos de construcción, y una matriz de requerimientos de distancias entre cada par de nodos terminales.

Se desea diseñar una subred 2-nodo conexa de costo mínimo, satisfaciendo las restricciones requeridas de largo de caminos entre todo par de nodos terminales. Este problema se denomina Diseño de Redes 2-Nodo-Conexas de Costo Mínimo con Nodos de Steiner y Caminos Acotados (2NCON-BP-SN, por sus siglas inglesas "2-Node CONnected Bounded Problem with Steiner Nodes").

El problema es intrínsecamente intratable del punto de vista computacional, por ser una generalización del Problema de Steiner 2-Nodo Conexo (STNSNP). Como consecuencia, se desarrolla una metaheurística GRASP, que ha mostrado gran aplicabilidad en problemas de telecomunicaciones.

En una primera etapa, la cual se denomina fase de construcción, se crean caminos iterativamente, con inspiración en el algoritmo CADILAC ("CAminos DIsjuntos de Largo Acotado").

A efectos comparativos, se ha definido una noción ávida o Greedy de resolución. Los resultados numéricos sobre instancias basadas en la librería TSPLIB indican un ahorro significativo con respecto a la solución Greedy.

Palabras Clave: **Optimización de Redes**, **Metaheurísticas**, **GRASP**, **CADILAC**

Introducción

Motivación

El diseño topológico de redes está intrínsecamente ligado a la Ingeniería, donde se debe construir una red robusta, garantizando un nivel de servicio y en algunas oportunidades de manera adicional mínimo costo. En un caso extremo, si se desea construir una red conexa a mínimo costo se obtiene una topología de árbol, que si bien conlleva ahorros no es tolerante a ninguna falla. Una medida natural de robustez de una red es su tolerancia ante fallas de uno o más enlaces o nodos. En el área de Telecomunicaciones, se exige usualmente 2-nodo conectividad, lo que implica que la red permanece operativa ante la falla de un solo componente cualquiera (enlace o nodo). Como consecuencia, la detección precoz y corrección de la falla lleva a que el servicio permanezca siempre en operación, salvo ante una simultánea falla de dos componentes, hecho que cae dentro de la categoría de evento raro.

Históricamente, las redes de comunicación presentan topologías de anillo, dada su simplicidad de construcción y robustez. El anillo es la topología que requiere menos enlaces para obtener una red 2-nodo conexa, e intuitivamente sugiere mínimo costo. No obstante, trabajos fundacionales en el diseño topológico de redes muestran que en casos especiales, existen otras redes de menor costo que son 2-nodo conexas pero no presentan topología de anillo.

Actualmente, las aplicaciones sensibles a la latencia como videoconferencia exigen que los terminales se deben comunicar con una cantidad limitadas de saltos. Como consecuencia, en este proyecto se estudia un problema de interés actual en conexto de telecomunicaciones. Dada una red con posibles enlaces y nodos intermedios a construir, se desea comunicar un conjunto de nodos terminales a mínimo costo en una subred 2-nodo conexa, donde tanto los enlaces como los nodos no terminales (denominados nodos de Steiner) tienen costo real positivo. El problema se denomina 2NCON-BP-SN por sus siglas inglesas, y pertenece a la clase de problemas NP-Difíciles, por ser una generalización del Problema del Viajero Ambulante o TSP. Como consecuencia, en este documento se describen metaheurísticas para abordar el 2NCON-BP-SN.

Este proyecto se focaliza en resolver el problema de conseguir para todo par de nodos 2 caminos nodo disjuntos entre ellos de costo mínimo con una cantidad máxima de hops a utilizar y con ciertos nodos llamados Nodos de Steiner que tendrán un costo extra de construcción, formando parte de algún camino nodo disjunto entre un par de nodos.

La red resultante contará con 2 caminos nodo disjuntos (esto quiere decir que no compartan ningún nodo) para todo par de nodos terminales que se elija. Estos nodos terminales son los nodos que se consideran distinguidos en la red, para los cuales debe existir 2 formas de comunicarlos y totalmente independientes.

La red original posee un conjunto de Nodos de Steiner, los cuales, si bien no son imprescindibles en la red ayudarán a dar conectividad entre los nodos terminales pero con un costo de construcción al utilizarlos. Por esta razón se tratará de no utilizar dichos nodos a menos que sea estrictamente necesario y la utilización de los mismos minimice el costo final de la solución.

Por otro lado también se tiene una máxima cantidad de hops para cada camino que une a un par de nodos terminales. Esto dificultará más aún el

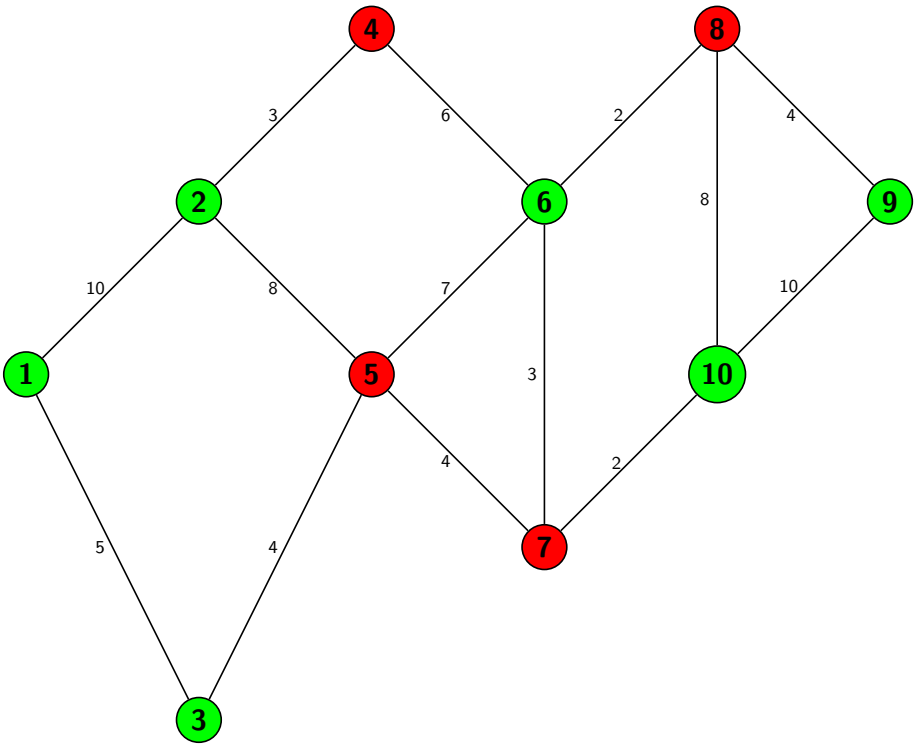

Figura 1: Grafo Simple no dirigido.

encontrar una solución que cumpla todos los requisitos y además, de bajo costo.

Definición formal del problema

Sea $G = (V, E)$, un grafo simple no dirigido como el de la Figura 1, el cual representa la red, con un conjunto V de vértices los cuales representan los nodos del problema y un conjunto E de aristas que representan las conexiones factibles del grafo.

Dentro del conjunto de vértices, existe un subconjunto $T \subseteq V$ de nodos que llamaremos terminales. Dichos nodos se encuentran de color verde en el grafo de la Figura 1. El resto de los nodos pertenecientes al conjunto de vértices son Nodos de Steiner, los cuales se encuentran de color rojo en el grafo anteriormente mencionado. Estos nodos nos ayudarán a poder construir la red buscada.

Existe también como entrada de este problema una matriz $D = \{d_{i,j}\}_{i,j \in T}$. Esta matriz indica la cantidad de hops máxima que puede tener cada camino entre un par de nodos terminales (nodos de color verde).

Por otro lado, existe una matriz $C = \{c_{i,j}\}_{(i,j) \in E}$, la cual tiene los valores de costo (enteros positivos) de los caminos (aristas) que unen los nodos (vértices) de la red (grafo).

Por último, existe un vector $C_2 = \{C_i\}_{i \in V \setminus T}$ el cual contiene los valores no negativos de construcción, que poseen los nodos que no son terminales (Nodos de Steiner). Es decir que en el vector C_2 se tiene los costos que se deben ir

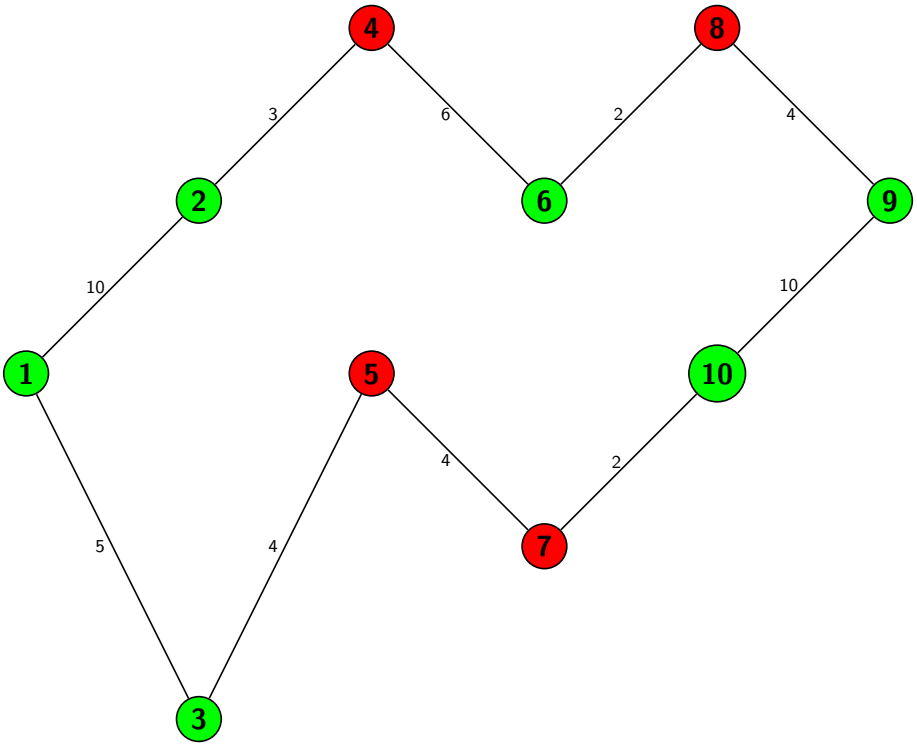

Figura 2: Solución obtenida luego de la ejecución de la metaheurística.

sumando al utilizar uno de estos nodos en nuestra solución.

Finalmente con todos estos parámetros de entrada el problema consiste en encontrar un subgrafo G', el cual sea de costo mínimo, que cumpla que para todo par de nodos terminales, existan dos caminos nodo disjuntos y que no excedan la cantidad máxima de hops, pasada por parámetro para cada par de nodos terminales.

Formalmente podemos plantearlo como encontrar $G' = (V', H), V' \subseteq V, T \subseteq V', H \subseteq E$, 2-nodo-conexo, de costo mínimo tal que $\forall i, j \in T, \exists P_{i,j} = \{i, v_1, v_2, \cdots, v_{n-1}, j\} \subseteq G'$ donde $n \leq d_{i,j}$ y $\exists Q_{i,j} = \{i, w_1, w_2, \cdots, w_{m-1}, j\} \subseteq G'$ donde $m \leq d_{i,j}$ disjuntos.

Una posible solución al grafo presentado en la Figura 1 es el presentado en la Figura 2

Complejidad Computacional del Problema

Para poder estudiar la complejidad computacional de este problema, se irá reduciendo el mismo y se probará que una simplificación del mismo es **NP-Difícil**.

Se realizarán unas simplificaciones al problema:

- Retirar todos los Nodos de Steiner
- Reducir todos los costos de las aristas a uno
- $d_{i,j} = \infty \forall (i, j) \in T$ conjunto de terminales

Una vez realizadas estas simplificaciones, ¿existe una solución factible con costo $\leq n$?

Si la respuesta a dicha pregunta es **SI**, entonces la solución es un ciclo Hamiltoniano ya que por la definición de ciclo Hamiltoniano es un ciclo que recorre todos los vértices del grafo. Si la respuesta fuese **NO** entonces no existe un ciclo Hamiltoniano.

El problema de decisión asociado al problema del Ciclo Hamiltoniano, se encuentra entre los problemas demostrados por Karp que son **NP-Completos**. Por lo tanto de esta manera, ya se prueba que el problema de decisión asociado a un sub-problema del estudiado en este proyecto, es **NP-Completo**, es decir que el problema de decisión del problema original, es **NP-Completo**. Además, se tiene que el problema es **NP-Difícil**, ya que un subproblema de él lo es, y por lo tanto el problema también lo es.

Organización del documento

El presente informe se encuentra dividido en 7 capítulos. El primero de ellos contiene las definiciones de varios conceptos relativos a Grafos, Algorítmica y Metaheurísticas y Complejidad Computacional que serán utilizados durante todo el mismo. Estos conceptos deben estar claros desde el comienzo para poder tener una fácil y amena lectura del presente trabajo.

En el Capítulo 2 se presenta un resumen de los trabajos más importantes en el área que se encuentran relacionados con el problema a resolver.

El problema que se resuelve en este trabajo está muy relacionado con otro problema ya estudiado y resuelto de manera muy eficiente, el cual se denomina CADILAC [1] debido a que resuelve el problema de caminos nodo disjuntos de largo acotado. Dicho trabajo se encuentra explicado detalladamente en el Capítulo 3 del presente informe.

En el Capítulo 4 se pasará a explicar en detalle cómo se resolvió el problema aquí presentado. En el mismo se dará explicación del algoritmo diseñado, usando la metodología GRASP (Greedy Randomized Adaptive Search Procedures) [2], la cual cuenta de dos fases. La primera de ellas, es una fase de construcción, en la cual se genera una solución factible del problema, y la segunda fase, en la cual se ejecutan las distintas búsquedas locales, que se desarrollaron para resolver el problema.

En el Capítulo 5 se presenta una solución golosa la cual se implementó a efectos de tener otra solución con la cual poder comparar la metaheurística aquí presentada. Además, se presenta una amplia tabla de resultados la cual posee los resultados obtenidos tanto en la metaheurística como en la solución golosa que se utilizó para comparar resultados.

En los dos últimos capítulos (Capítulo 6 y 7) se presentan las conclusiones y el trabajo futuro de este proyecto.

Índice

1. Capítulo 1 - Terminología

1.1. Grafos

Se brindan a continuación conceptos relativos a teoría de grafos con el fin de facilitar la comprensión de este documento. Todas ellas fueron extraídas de Design of Survivable Networks [3], Graph Theory [4] y Heuristic Methods for the Hop Constrained Survivable Network Design Problem [5].

Definición 1. *Dado un grafo G=(V,E), se denota |G| y ‖ G ‖ a la cantidad de nodos y aristas de G respectivamente. Así como también | V | denota la cantidad de Vértices y |E| la cantidad de aristas.*

Definición 2. *Un grafo completo es un grafo simple donde cada par de vértices está conectado mediante una arista. Un grafo completo de n nodos se denota como K_n.*

Definición 3. *Un grafo es métrico si se encuentra definida una función $d : X^2 \to R_+$ que cumple las siguientes condiciones:*

- *Desigualdad del triángulo:* $\forall a, b, c \in X d(a, b) \leq d(a, c) + d(c, b)$.
- *Propiedad simétrica:* $\forall a, b \in X d(a, b) = d(b, a)$.
- $\forall a \in X d(a, a) = 0$.
- $\forall a, b \in X$, *si* $d(a, b) = 0$, *entonces* $a = b$.

Definición 4. *Dado un grafo $G = (V, E)$ y dos subconjuntos de nodos $X, Y \subseteq V$, se denota con $E(X, Y)$ al conjunto de aristas de E que tienen un extremo en X y otro en Y.*

Definición 5. *Un grafo G=(V,E) es* ***k-arista-conexo*** *si $|V| > k$ y $G \backslash F$ es conexo para todo $F \subseteq E$ con $|F| < k$. Por lo cual, ningún par de nodos de G están separados por menos de k aristas.*

Definición 6. *El mayor valor de k para el cual el grafo G es* ***k-arista-conexo*** *se le llama arista-conectividad de G, denotada $\lambda(G)$.*

Definición 7. ***Componente conexa:*** *Una componente conexa de un grafo es un subgrafo conexo maximal.*

Definición 8. *Sea $G = (V, E)$ un grafo, se dice que un nodo $v \in V$ es un* ***punto de articulación*** *si separa a dos nodos de la misma componente i.e. $G \backslash v$ es desconexo.*

Definición 9. *Sea $G = (V, E)$ un grafo, se dice que una arista $e \in G$ es un* ***puente*** *si separa a dos nodos de la misma componente; i.e. $G \backslash e$ es desconexo.*

Definición 10. *Un grafo $G = (V, E)$ es* ***k-nodo-conexo*** *si $|V| > k$ y $G \backslash X$ es conexo para todo $X \subseteq V$ con $|X| < k$. Por lo cual, ningún par de nodos de G están separados por menos de k nodos.*

Definición 11. *El mayor valor de k para el cual un grafo G es* ***k-nodo-conexo*** *se le llama conectividad de G, denotada $K(G)$.*

Definición 12. ***k-nodo-sobreviviencia:*** *Un grafo $G = (V, E)$ se dice* ***k-nodo-sobrevivible*** *si entre cualquier par de nodos s y t existe entre ellos al menos k caminos nodo-disjuntos, donde aristas paralelas entre s y t son contadas como caminos nodo-disjuntos diferentes.*

Definición 13. ***Problema de optimización combinatoria:*** *Un problema de optimización combinatoria es de la forma $min_{x \in D} f(x)$, siendo D un conjunto finito.*

Definición 14. ***SNDP (Survivable Network Design Problem):*** *Es un importante conjunto de problemas algorítmicos en los que su objetivo es encontrar la red de mínimo costo para la cual se cumple alguna propiedad de conectividad deseada.*

Definición 15. ***MW2CSN:*** *Este problema consiste en encontrar el subconjunto de costo mínimo de aristas de un grafo tal que cualquier par de nodos posee 2 caminos nodo disjuntos.*

1.2. Complejidad Computacional

1.2.1. ¿Qué es el conjunto NP?

El lector puede hallar un valioso texto introductorio a la complejidad computacional en *Computers and Intractability, A Guide to the Theory of NP - Completeness* [6]. A continuación se brindarán sus conceptos fundamentales.

Definición 16. *Un* ***algoritmo eficiente*** *es un algoritmo de complejidad polinomial, así como también se dirá que un problema se encuentra* ***correctamente resuelto*** *si se conoce algoritmos eficientes para resolverlo.*

La idea principal es la de poder clasificar los problemas según su complejidad.

Definición 17. *Los* ***problemas de decisión*** *reciben como entrada una cierta cantidad de parámetros y tienen como salida una respuesta $\mathcal{SI}$ o $\mathcal{NO}$.*

Definición 18. *Las* ***instancias de un problema*** *son una especificación de sus parámetros. Por lo tanto un problema de decisión π tiene asociado un conjunto D_π de instancias y un subconjunto $Y_\pi \subseteq D_\pi$ de instancias cuya respuesta es $\mathcal{SI}$.*

Existen diferentes versiones de un problema de optimización π. De esta manera si se tiene una instancia **I** del problema π se tiene:

- **Evaluación:** Se determina el valor óptimo de π para **I**
- **Optimización:** Se encuentra una solución óptima del problema π para **I** (de valor mínimo o máximo)
- **Decisión:** Si se fija un número K, ¿existe una solución factible de π para **I** tal que $c(S) \leq K$ si el problema es de minimización o $c(S) \geq K$ si el problema es de maximización?
- **Localización:** Si se fija un número K, determinar una solución factible de π para **I** tal que $c(S) \leq K$.

Existen problemas considerados **intratables** que son aquellos que no pueden ser resueltos por un algoritmo eficiente. Las causas de que un problema entre en esta categoría pueden ser varias:

- El problema en cuestión necesita de una respuesta de longitud exponencial. (Ejemplo: Todos los circuitos hamiltonianos de longitud a lo sumo k).
- El problema es indecidible (Ejemplo: Problema de la parada, planteado en *On Computable Numbers, with an Application to the Entscheidungsproblem* [7]).
- El problema es decidible pero no se conoce algoritmos polinomiales que sean capaces de resolverlo.

Definición 19. *Una **Máquina de Turing** es un dispositivo que manipula símbolos sobre una tira de cinta de acuerdo a una tabla de reglas y si tenemos que para cada entrada tiene una única salida posible se le denomina **Máquina de Turing determinística**. En caso contrario se le denomina **Máquina de Turing no determinística**.*

Para las Máquinas de Turing determinísticas (**DTM**) se dice que una máquina $\mathcal{M}$ resuelve un problema π si para toda instancia empieza, termina y contesta bien o lo que es lo mismo termina en un estado final correcto.

La complejidad de estas máquinas viene dada por la cantidad de movimientos de la cabeza desde el estado inicial hasta alcanzar un estado final en función del tamaño de la entrada: $\mathcal{T}_M$(n) = máx { m tq $\exists$ x $\in \mathcal{D}_\pi$, | x | = n y M con input x ejecuta m movimientos}.

Definición 20. *Un problema se encuentra en el conjunto $\mathcal{P}$ si existe una **Máquina de Turing Determinística** de complejidad polinomial que lo resuelve. $\mathcal{P}$ = máx { π tq $\exists$ DTM tq M resuelve π y $T_M(n) \in O(p(n))$ para algún polinomio p }.*

Por otro lado si se observa las máquinas no determinísticas (**NDTM**) se dice que es polinomial para π si existe una función poinomial $T(n)$ de manera que para toda instancia de $\mathcal{Y}_\pi$ de tamaño n, algunas de las ramas termina en estado $\mathcal{Q}_{si}$ en a lo sumo T(n) pasos.

Definición 21. ***un problema $\pi \in \mathcal{NP}$ si existe una NDTM polinomial que resuelve π***

De forma equivalente se tiene que un problema de decisión pertenece a la clase $\mathcal{NP}$ si dada una instancia de $\mathcal{SI}$ y evidencia de la misma se puede verificar en tiempo polinomial. Dicha evidencia se llama en algunos casos **certificado** y tiene que tener tamaño polinomial en el tamaño de la entrada.

1.2.2. ¿Qué es NP-Completo?

Para poder contestar la pregunta aquí planteada se ncesita definir previamente el concepto de reducción polinomial.

Definición 22. ***Reducción polinomial:*** *Es una manera que tenemos de relacionar dos problemas de decisión de forma que la existencia de un algoritmo que resuelve el primer problema, garantiza inmediatamente, y a través de un tiempo polinómico, la existencia de un algoritmo que resuelve el segundo.*

Con esta definición en mente, se tiene que un problema π forma parte de la clase **NP-Completo** si cumple dos condiciones:

1. $\pi \in \mathcal{NP}$.
2. Para todo π' $\in \mathcal{NP}$, π' reduce polinomialmente a π.

1.2.3. ¿Qué es NP-Difícil?

Definición 23. *Un problema de decisión π es NP-Difícil si todo problema de $\mathcal{NP}$ se puede transformar polinomialmente a π. Éste conjunto es aquel que contiene todos los problemas de decisión que son como mínimo tan difíciles como un problema de NP.*

1.3. Algorítmica y Metaheurísticas

1.3.1. Introducción a Metaheurísticas

Heurística proviene del griego heuriskein y significa "encontrar" o "descubrir". Las heurísticas son técnicas que se centran en buscar soluciones de buena calidad y a un costo computacional bueno, pero no garantiza la optimalidad de dicha solución, o lo que es peor es que ni se conoce su grado de error.

Las heurísticas se pueden clasificar de la siguiente manera:

- **Métodos Constructivos:** Estos métodos generan soluciones partiendo desde una solución vacía y luego agregan componentes hasta completar la solución.
- **Métodos de Búsqueda Local:** Estos métodos parten desde una solución y de forma iterativa reemplazan la solución actual con una similar pero de mayor calidad.

Las **metaheurísticas** cuentan de los siguientes objetivos:

- Encuentran soluciones factibles y de buena calidad, cuyo valor es cercano al óptimo.
- Encuentran de manera rápida soluciones factibles, en problemas **NP-Dificil**.
- Estas metaheurísticas recorren el espacio de soluciones de manera de no quedar "estancados" en una zona.

Las metaheurísticas pueden clasificarse de diferente manera, sin ser exhaustivos se presenta algunas de ellas:

1. **Provenientes de la Naturaleza:**

- **Algoritmos Genéticos** - Evolución de las Especies.
- **Colonias de Hormigas** - Enfriamiento de Metales.
- **Búsqueda Dispersa** - no bio-inspirada.

2. **Aleatorias vs Determinísticas**

   - **Aleatorias** - Algoritmos Genéticos, Colonias de Hormigas.
   - **Determinísticas** - Búsqueda Tabú y Búsqueda Dispersa.

3. **Inspiradas en un individuo vs Inspiradas en Poblaciones**

   - **Inspiradas en un individuo** - GRASP, Búsqueda Tabú, Búsqueda Local y variantes.
   - **Inspiradas en Poblaciones** - Algoritmos Genéticos, Colonias de Hormigas, Búsqueda Dispersa.

4. **Con Memoria vs Sin Memoria**

   - **Sin Memoria** - GRASP, Búsqueda Local y variantes.
   - **Con Memoria** - Búsqueda Tabú y Colonias de Hormigas.

1.3.2. GRASP

GRASP (Greedy Randomized Adaptive Search Procedures) es una metaheurística de gran efectividad y capaz de obtener las mejores soluciones a varios problemas presentados en *Essays and surveys in Metaheuristics* [8]. Esta metaheurística fue por primera vez presentada en *Greedy Randomized Adaptive Search Procedures* [2].

Esta metaheurística es iterativa y contiene dos fases bien diferenciadas las cuales son repetidas un número predeterminado de veces y son llamadas *Fase de Construcción* y *Búsqueda Local*.

La primera de ellas consiste en obtener una solución factible mediante un algoritmo greedy aleatorizado.

Por otro lado, la segunda fase es la encargada de recorrer una estructura de vecindad definida sobre una solución factible buscando un óptimo local.

En el procedimiento denominado GRASP (Figura 3) aquí presentado se puede observar la estructura general de la metaheurística. En dicho procedimiento *NúmeroIteraciones* es la cantidad de iteraciones realizadas por el algoritmo. *TamañoLista* es el tamaño utilizado por la lista de candidatos utilizada en la *Fase de Construcción* y *Semilla* es la semilla utilizada por el generador de números pseudoaleatorios.

Fase de Construcción del GRASP

En esta fase se construye una solución factible de forma golosa y aleatorizada. Se comienza con una lista vacía y se agrega elementos a partir de una lista de candidatos denominada *RCL (Restricted Candidate List)*, en cada paso. La *RCL* tiene una cantidad de *TamañoLista* elementos, los cuales pueden ser agregados en cada paso, elegidos por una función que es la encargada de con la información

Algoritmo 1 *GRASP*(*NúmeroIteraciones*, *TamañoLista*, *Semilla*)

```
1: f* ← ∞
2: for k = 1..NúmeroIteraciones do
3:    S ← AlgoritmoGolosoAleatorio(TamañoLista, Semilla)
4:    if f(S) < f(S') then
5:       S* ← S;
6:       f* ← f(S);
7:    end if
8: end for
9: return S*
```

Figura 3: Pseudocódigo del método GRASP.

Algoritmo 2 *AlgoritmoGolosoAleatorio*(*TamañoLista*, *Semilla*)

```
1: //TamañoLista = tamaño máximo para la lista restringida de candidatos.
2: //Semilla = generador de números seudoaleatorios.
3: //El conjunto E contiene los elementos que pueden ser agregados incrementalmente a la solución parcial S.
4: S ← ∅;
5: //Evaluar el costo incremental de cada e ∈ E
6: while not EsFactible(S) do
7:    RCL ← GenerarRCL(TamañoLista);
8:    s ← SeleccionarElementoAleatorio(RCL);
9:    S ← S ∪ {s};
10:   //Actualizar costo incremental de cada e ∈ E\S
11: end while
12: return S;
```

Figura 4: Pseudocódigo del método de AlgoritmoGolosoAleatorio.

disponible, calcular el beneficio de agregarlos en ese momento. En la mayoría de los casos se elige el elemento que produce un mayor beneficio en el costo en la solución parcial, pero igual otras estrategias son permitidas. Para elegir el elemento perteneciente a la *RCL* que se desea agregar a la solución se sortea de forma aleatoria. Todo lo detallado anteriormente se repite hasta que se obtenga una solución factible para el problema.

Fase de Búsqueda Local del GRASP

En la mayoría de los casos las soluciones encontradas por la Fase de Construcción no resultan ser óptimas y para mejorarlas se aplica una *búsqueda local*. En esta fase se mejora la solución factible S encontrada en la fase previa del *GRASP* buscando dentro de una vecindad $N(S)$ definida para esta utilidad. La efectividad de esta etapa depende de:

Algoritmo 3 *BusquedaLocal*(*NúmeroIteraciones*, *TamañoLista*, *Semilla*)

```
1: while not EsOptimoLocal(S) do
2:    Buscar S' ∈ N(S) tal que f(S') < f(S)
3:    if Existe(S') then
4:       S ← S';
5:    end if
6: end while
7: return S;
```

Figura 5: Pseudocódigo del método de BúsquedaLocal.

- Estructura de la Vecindad.
- Estrategia de búsqueda dentro de dicha vecindad.
- La velocidad de la función de la evaluación de costo.
- Solución inicial.

Además, existen dos estrategias de búsqueda dentro de la vecindad las cuales se encuentran descritas en *Greedy randomized adaptive search procedures* [9]:

1. *Best-improving:* Se visitan todos los elementos del vecindario $N(S)$ de la solución S y tomamos el mejor de ellos si alguno de los mismos logra superar a S
2. *First-improving:* Se reemplaza la solución S por el primer vecino que mejora su costo.

1.3.3. VNS y VND

VNS (Variable Neighborhood search) [10] es una metaheurística que permite resolver un Problema de Optimización Combinatoria y optimización global de problemas. Esta técnica investiga los vecindarios más alejados de la solución actual (la encontrada hasta el momento) y se mueve de allí sí y solo sí se obtiene una mejora. El método de búsqueda local se aplica varias veces para ir de soluciones obtenidas en el vecindario a óptimos locales. VNS está diseñado para optimizar las soluciones encontradas para problemas de optimización discreta y continua por lo cual se encuentra dirigido a resolver problemas de programación lineal, problemas de programación entera, problemas de programación entera mixta, problemas de programación no lineal, entre otros. VND (Variable Neighborhood Descent) [11] se obtiene si se realiza un cambio en las vecindades en una forma determinística. En las descripciones de sus soluciones se asume que se tiene una solución inicial. La mayoría de las heurísticas de búsqueda local utilizan muy pocas vecindades. La solución final debe ser un mínimo local con respecto a todos los $\mathcal{K}_{\mathbf{max}}$ de las vecindades. Por lo tanto con VND se tiene más posibilidades de llegar a un óptimo global que utilizando una sola estructura de vecindad.

2. Capítulo 2 - Trabajos Relacionados

2.1. Trabajos Relacionados

Tradicionalmente la disponibilidad ha sido la mayor causa de preocupación en los servicios telefónicos. Un árbol provee disponibilidad y puede ser eficientemente diseñado por Kruskal en su algoritmo en 1956 [12], pero no es suficientemente robusto ya que ante una sola falla en alguno de sus nodos, la red en cuestión quedará disconexa. Los nodos terminales deberían estar conectados por uno o más caminos donde los nodos de Steiner ayudan a brindar robustez en nuestra red.

La teoría de complejidad computacional muestra que encontrar el árbol de Steiner de mínimo costo es una complicada tarea [13]. Distintas aplicaciones pueden encontrarse en la tesis de Robledo [14]. La sobrevivilidad debe ser tenida en cuenta en un sistema que se encuentra expuesto a potenciales fallas como por ejemplo comunicaciones a través de fibra óptica. El lector podrá encontrar un amplio desarrollo del tema en el libro Mechthild Stoer, que contiene una gran cantidad de resultados de diseño de redes sobrevivibles [3], así como también el trabajo de Kerivin y Mahjoub [15].

Una red dos nodo conectada sigue conectada ante la pérdida de uno de sus nodos. Pensando en el costo-beneficio en el diseño de redes, son ampliamente consideradas las redes que poseen topologías dos nodo conexas para proveer robustez ante caídas de un único nodo de la red.

En telecomunicaciones un método tradicional multiterminales es usar anillos. El objetivo del Capacitated m Ring Tree Problem (CmRSP) es el de conectar terminales por m anillos con un nodo distinguido y posiblemente algunos nodos colgantes a mínimo costo. Una red mínimamente conectada provee disponibilidad pero no robustez ante una simple falla de uno de sus nodos o links. En las comunicaciones de fibra óptica, la robustez es algo esencial por lo cual las redes dos nodo conexas son de gran importancia. Un acercamiento natural a la dos nodo conectividad es conectar todos los terminales mediante un anillo o ciclo de una manera económica. De esta manera cada nodo se encuentra conectado a los demás mediante dos caminos nodo disjuntos. Este problema denominado TSP es ampliamente estudiado por la literatura científica. [16].

Un resultado fundamental es atribuido a Monma et. al. [17]. Allí los autores estudiaron el problema de Minimum Weight Two Connected Spanning Problem (MW2CSP), donde el objetivo es encontrar el subgrafo de menor costo, dos nodo conectado el cual es un grafo métrico.

En dicho paper se logra probar que existen mejores soluciones que los anillos en ciertos escenarios. Sin embargo, el costo del mejor anillo no puede exceder el costo de la mejor solución en 4/3. Un ejemplo de esto se puede ver en el siguiente ejemplo.

Sea el grafo completo K_5 de cinco nodos presentado en la Figura 6. Todas las aristas de costo uno forman un grafo dos nodo conexo (M), denominado grafo de Monma, y tiene un costo de $c(M) = 6$ representado en la Figura 7.

Sin embargo, cualquier anillo deberá usar una arista de las de costo 10, por lo cual el costo del anillo de mínimo costo C, es $c(C) = 14$. En el ejemplo se tiene que $c(C)/c(M) > 4/3$, y el costo no cumple con la desigualdad triangular

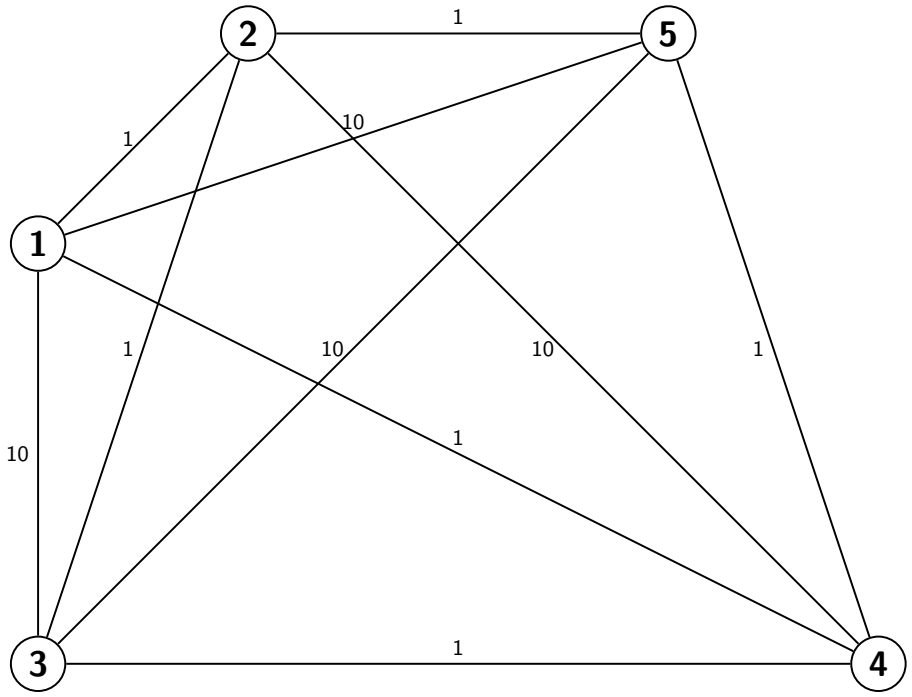

Figura 6: k_5. Grafo completo de cinco nodos.

o lo que es lo mismo, la hipótesis de grafo métrico no se cumple.

Por otro lado se tiene el Ring Star Problem (RSP), el cual fue introducido por Labbé et. al. [18] inspirado en el diseño de redes de fibra óptica. Es una generalización del problema del camino Hamiltoniano, en donde algunos nodos terminales deben ser conectados directamente a un anillo. El término "star" (estrella) se desprende por esos nodos "hoja", los cuales se encuentran conectados al anillo.

Formalmente, se tiene un grafo $G = (V, E \cup A)$ donde E representa las aristas no dirigidas con costos $\{c_e\}_{e \in E}$ y A representa arcos dirigidos con costos $\{d_e\}_{e \in A}$. Existe un nodo distinguido $v_1 \in V$, denominado depot. El objetivo del problema es diseñar un subgrafo de cubrimiento $G' = C \cup T$, siendo $C \subseteq E$ un ciclo que incluye a el depot v_1 y $T \subseteq A$ directamente conectado por arcos saliendo del ciclo, donde el costo global $\sum_{e \in C} c_e + \sum_{e \in A} d_e$ es minimizado.

El problema RSP pertenece a la clase $NP - Hard$ y existen soluciones exactas para ciertas instancias [19]. Este problema es utilizado en una amplia gama de aplicaciones, entre ellas se destacan: Redes de Comunicación, logística, ubicación de contenedores de basura y transporte. Por esta razón es que gran parte del trabajo aplicado a este problema, suele utilizar métodos exactos y de aproximación.

En 2004 Martín Labbé et. al. diseñó por primera vez un algoritmo utilizando la técnica de ramificado y acotamiento capaz de resolver instancias de 200 nodos [20]. Hasta donde se conoce este es el primer método exacto disponible en la literatura. Los autores incluyen tres clases de instancias de problemas tomados de la biblioteca TSPLIB llegando en muchos de ellos al valor óptimo.

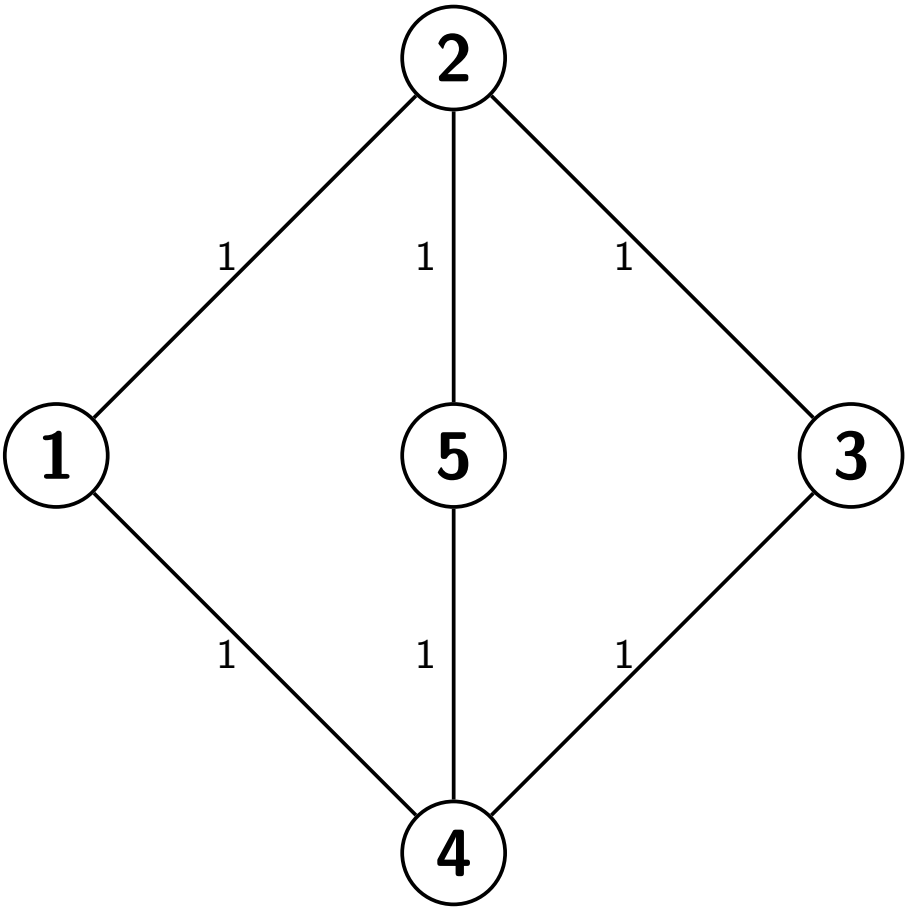

Figura 7: Grafo de Monma más barato que un anillo.

Otra generalización de dicho problema fue realizada por Roberto Baldacci et. al centrado en soluciones realistas donde los consumidores se encuentran geográficamente distribuidos [21]. En ese trabajo los autores consideran m bloques donde solamente tienen al depot como nodo común. Estos bloques son nuevamente anillos y su principal diferencia con el RSP es la presencia de m anillos en vez de solo uno. Ambos problemas pertenecen a la clase NP-Hard debido a que son una generalización del problema del Camino Hamiltoniano [13]. Un intento de equilibrio entre la robustez y el costo es propuesto por Allessandro Hill et. al [22]. En dicho trabajo existe en el centro nuevamente un anillo pero existen ciertos nodos conectados al anillo mediante árboles. El problema obtenido es denominado Capacitated Ring-Tree Problem o CRTP en su manera abreviada.

S. Dias et. al. presentó una metodología GRASP para el RSP [23]. Se agrega randomicidad durante la búsqueda local en la cual el mejor candidato no es agregado pero una lista de candidatos (RCL) es utilizada. Agregar/quitar y k-opt son búsquedas locales implementadas en GVNS (General Variable Neighborhood Search). Este trabajo también fue testeado utilizando instancias de la biblioteca TSPLIB. Kedad-Sidhoum et. al. propusieron un método exacto utilizando ramificado y acotamiento resolviendo una nueva formulación matemática del RSP, la cual es basada en cadenas s-t [19]. Kedad-Sidhoum et. al. testearon sobre las mismas instancias que Labbé et. al. en 2004 y pudieron hallar la mejor solución 15 veces más rápido, además de obtener la mejor solución para la instancia Kroa200. En 2013 H.Calvete et. al. desarrolló un algoritmo evolutivo para resolver el RSP basándose en una formulación en dos niveles [24]. Aquí se consideraron instancias de la TSPLIB entre 50 y 200 nodos y se concluyó que este nuevo algoritmo presentó las mejores soluciones hasta el momento para el 92,74 % de los casos.

Trabajos recientes en el diseño estructural de redes reemplaza los anillos por componentes dos nodo conexas arbitrarias inspirados en los abaratamientos de costos previstos por Clyde Monma et. al. Un ejemplo de ello es el trabajo de Gabriel Bayá et. al que introdujo el Capacitated m Two-Node Survivable Star Problem, o en su manera abreviada CmTNSSP [25]. Bayá reemplaza los m anillos del CmRSP por dos componentes dos nodo conexas. Análogamente Rodrigo Recoba et. al. presenta el problema Two-Node Connected Star Problem (TNCSP) el cual consiste en el problema RSP pero con una componente dos nodo conexa reemplazando el anillo [26]. En ese paper, se presenta una extensión natural del CRTP y el CmRSP, en donde se tiene m bloques dos nodo conectados y en los puntos de acceso de la red árboles conectados a dichos bloques. El objetivo es lograr mayor flexibilidad y ahorro de costo simultáneamente.

2.2. Teoremas Relevantes del Área

En esta sección se encuentran resumidos los teoremas más importantes citados a lo largo de este documento. Elementos teóricos principalmente de la teoría de grafos, la complejidad computacional y metaheurísticas.

Teorema 1. ***Versión global del Teorema de Menger:***

1. *Un grafo es k-nodo-conexo sii contiene k caminos independientes (nodo disjuntos) entre cualquier par de nodos.*

2. *Un grafo es k-arista-conexo sii contiene k caminos arista disjuntos entre cualquier par de nodos.*

Teorema 2. *Sea un grafo métrico, para cualquier conjunto de vértices con función de distancia d(.) definida en $\mathcal{V} \times \mathcal{V}$ existe un grafo $G = (V, E)$ 2-conexo de costo mínimo que satisface las siguientes condiciones:*

1. *Todo vértice en G tiene grado 2 o 3.*
2. *Si removemos una o dos aristas en G obtendremos una arista puente en alguna de las componentes conexas resultantes de G.*

Demostración: Esta demostración puede encontrarse en *Minimum-weight two-connected spanning networks* [17].

Teorema 3. ***Berge 1973:*** *Sea G = (V,E) un grafo que satisface las siguientes condiciones:*

1. *G es 2-conexo.*
2. *Todo vértice de G tiene grado 2 o 3.*
3. *G es arista minimal.*
4. *Removiendo un par de aristas en G deja una arista puente en una de las componentes conexas resultantes.*

Se cumple entonces que G es el único grafo 2-conexo de cubrimiento de costo mínimo para la función de distancia canónica.

Demostración: Esta demostración puede encontrarse en [17].

Teorema 4. *Según el lema 2 [17] tenemos que dado $G = (V, E)$ un grafo 2-conexo con $G'(V', E')$ un subgrafo inducido por V'. Luego reemplazando E' en G por cualquier colección de aristas de E'' definidas en V', donde $G'' = (V', E'')$ es 2-conexo, resulta en un grafo $G^* = (V, (E \backslash E') \cup E'')$ el cual es 2-conexo.*

3. Capítulo 3 - CADILAC

Como componente importante del algoritmo que resuelve el problema abordado en este trabajo, se encuentra el algoritmo CADILAC [1].

Este algoritmo permite hallar k caminos nodo-disjunto de largo acotado entre dos terminales de una red. Esta red tiene costos en sus aristas, y la construcción de tales caminos se debe realizar a costo mínimo, sujeto a la condición de que cada camino debe tener un largo d o inferior. Este problema pertenece a la clase de computacional de problemas $NP - Completos$ y por lo tanto no tiene solución polinomial conocida.

CADILAC sigue la metaheurística GRASP, agrega aleatorización y diversidad adaptativa a una resolución golosa inspirada en el algoritmo de Bhandari y del algoritmo DIMCRA, los cuales son explicados en secciones posteriores.

El problema resuelto por CADILAC es de gran importancia en varios contextos. El problema en su variante más básica, en donde las aristas del grafo tienen costos reales no negativos y se busca minimizar el costo total de la solución, no imponiendo mayores restricciones sobre los caminos además de que sean disjuntos, tienen soluciones polinomiales conocidas [27], [28], [29]. Por otro lado, en otras aplicaciones suelen aparecer restricciones adicionales que vuelven al problema computacionalmente más complejo. Un ejemplo de esto es cuando Guo et. al. [30] muestran que el problema de hallar dos caminos arista disjuntos entre un par de nodos de un grafo, en el cual las aristas poseen costos multidimensionales y cada camino de la solución debe obedecer a un vector de restricciones, es un problema $NP - Completo$.

El estudio de la complejidad computacional del problema que consiste en dados dos nodos, hallar la máxima cantidad de caminos nodo disjuntos con largo acotado entre dichos nodos se encuentra en Itai [31].

Formalizando CADILAC, se tiene un grafo G cuyas aristas tienen costo positivo y dos nodos s y t pertenecientes a G, buscando la solución de costo mínimo al problema de hallar k caminos simples nodo-disjuntos entre dichos nodos, con la restricción de que cada camino tenga a lo sumo d aristas.

Existen varios trabajos previos al problema de hallar k caminos nodo o arista disjuntos entre dos nodos de un grafo [32], [27], [33], [34], [35], [36], [37], [30]. Este problema en su formulación más sencilla tiene una solución conocida y en muchos casos se tratan variantes que imponen restricciones adicionales sobre los caminos, las cuales transforman el problema original a uno de mayor complejidad computacional.

Uno de los primeros autores en estudiar este problema y brindar una solución al mismo en su versión básica, para hallar caminos arista-disjuntos fue Suurballe [36], la cual se detalla en la siguiente sección.

3.1. Suurballe

El algoritmo de Suurballe [36] tiene como idea principal utilizar el algoritmo de Dijkstra para hallar el camino más corto entre los nodos de origen y destino, modificar los costos de las aristas sobre dicho camino, y luego correr el algoritmo de Dijkstra por segunda vez en el grafo resultante. Suurballe combina ambos caminos hallados para obtener dos caminos independientes que cumplen con las

Algoritmo 4 *Bhandari*$(g : Grafo, c : E \to R^+, k)$

1: *Solución* $\leftarrow \emptyset$
2: **while** $k > 0$ **do**
3: // Se crea un grafo G' ...
4: $G' \leftarrow$ Grafo()
5: // ... que comparte vértices con G ...
6: $V(G') \leftarrow V(G)$
7: // ... y sus aristas salvo aquellas que pertenecen ...
8: // también a *Solución* ...
9: $E(G') \leftarrow E(G) \setminus$ Solución $\cup \{(u, v) : (v, u) \in$ Solución $\}$
10: // ... y el costo de cada arista, a menos que
11: // ... pertenezcan a *Solución*, en cuyo caso se le asigna
12: // ... el costo opuesto.
13: $c' : E \to R^+$, $c'(e) = \{c(e)\ \forall e \notin$ *Solución*, $-c(e)\ \forall e \in$ *Solución*$\}$
14: ... Se halla el camino más corto entre *inicio*
15: ... y Fin en el grafo G' El algoritmo usado
16: ... es el de Dijkstra con la variante de que
17: ... los nodos pueden ser visitados varias veces.
18: $SP' \leftarrow$ *CaminoMásCorto(Inicio,Fin,G')*
19: Se combinan las aristas del camino hallado
20: ... con la solución parcial, eliminando las
21: ... aristas comunes a ambos conjuntos.
22: Solución $\leftarrow$ (Solución $\cup\ SP'$) $\setminus$ (Solución $\cap\ SP'$)
23: $k \leftarrow k - 1$
24: **end while**
25: **return** *Solución*

Figura 8: Pseudocódigo del método de Bhandari.

condiciones del problema. Luego, unos años más tarde se presenta una variación del algoritmo anterior [37], el cual logra un mejor orden de tiempo de ejecución: $\mathcal{O}(mlog_{(1+m/n)}n)$ frente al $\mathcal{O}(n^2\, log\, n)$ que se tenía en el algoritmo anterior para el caso $K = 2$.

3.2. Bhandari

Bhandari [27] propone un algoritmo que aborda el mismo problema que el algoritmo de Suurballe pero modificado. El mismo se encuentra explicado en la Figura 8.

3.3. DIMCRA

Kobayashi y Sommer [34] estudian el problema de hallar k caminos nodo-disjuntos con $k \in \{2, 3\}$, entre pares de nodos s_i, t_i para $i = 1..k$ de un grafo plano y no dirigido, minimizando dos funciones distintas: la suma total de los caminos y el largo máximo de los caminos. El problema aquí es más general

debido a que considera múltiples pares de nodos origen y destino los cuales son unidos por cada uno de los caminos de la solución. Este artículo se basa principalmente en la planaridad del grafo por lo que no son fácilmente aplicables a otros contextos como el del problema del estudio.

Fleischer et al. [35] abordan el problema de hallar k caminos arista o nodo-disjuntos entre dos nodos de un grafo acíclico (DAG por su sigla en ingles) con distintos objetivos: MinMax k-DP, Balanced k-DP, MinSum-MinMax k-DP y MinSum-MinMin k-DP. El problema MinMax k-DP busca una solución la cual minimiza el costo del camino más costoso. El problema Balanced k-DP por otro lado tiene como objetivo minimizar la diferencia de costos entre el camino de mayor costo y el de menor costo.

El MinSum-MinMax k-DP minimiza el costo total de la solución y adicionalmente dentro del conjunto de soluciones de costo mínimo, busca como segundo objetivo minimizar el costo de su camino más costoso. Por último tenemos el MinSum-MinMin k-DP, es similar al anterior pero tiene un segundo objetivo diferente, el cual consiste en minimizar el costo del camino menos costoso. Su algoritmo se basa en propuesto por Perl-Shiloach [35] la cual es aplicable solamente a grafos acíclicos dirigidos, donde se genera un grafo intermedio y luego buscan un camino de origen a destino en él.

En el trabajo de Guo et. al. [30] se propone una solución al problema de hallar dos caminos arista-disjuntos entre un par de nodos cuyo costo total sea mínimo. Este problema tiene como variante sobre el que ataca el algoritmo de Bhandari, que reside en que la función de costo no es lineal y en que cada camino debe obedecer a un vector de componentes w_m. El costo total de un camino P está definido como:

$$c(P) = max_{m=1..n}(\frac{w_m(P)}{C_m}) \tag{1}$$

donde

$$w_m(P) = \sum e \in P w_m(e) \tag{2}$$

y C es el vector de restricciones. La no linealidad de la función de costo produce que la siguiente propiedad no se cumpla, pero en el caso lineal sí se cumple.

Dado el camino más corto P_1 entre dos nodos s y t; si tomamos dos nodos u, v pertenecientes al camino, el fragmento del camino P_1 que une a u a con v, es el camino más corto entre u y v. Como consecuencia de la propiedad anterior en el algoritmo de Bhandari no se forman ciclos negativos, los cuales pueden provocar un bucle infinito en la ejecución. Además permite que el algoritmo de Dijkstra halle eficientemente el camino más corto entre dos nodos. En el caso no lineal no se cumple esta propiedad, por lo cual no se puede utilizar el algoritmo de Bhandari directamente para resolver el problema. Es por esta razón que los autores diseñan un nuevo algoritmo, DIMCRA, el cual se encuentra basado en el algoritmo de Bhandari.

A diferencia de Bhandari, DIMCRA es un algoritmo aproximado y al cambiar la dirección de las aristas del camino más corto, los costos no se cambian por

su valor opuesto sino que se establecen en 0. En cada paso, el camino más corto no se halla usando el algoritmo de Dijkstra, sino que en su lugar se utiliza un algoritmo denominado SAMCRA [38] y se considera un vector de restricciones $C' = 2C$.

Por último, si luego de combinar ambos caminos, alguno de los resultantes no cumple con el vector de restricciones C, las aristas del segundo camino hallado por SAMCRA que no se superponen con el camino más corto se eliminan del grafo, para luego volver a ejecutar la búsqueda. Lo más atractivo de este algoritmo para el problema que resuelve CADILAC es que puede ser utilizado directamente para resolver el caso donde $K = 2$, realizando una modificación sobre la función de costo.

Otro avance fue realizado por Xiong et al [33] donde profundizan aún más sobre el algoritmo de DIMCRA. Allí se propone una versión exacta del algoritmo, la cual mantiene el esquema básico. Los resultados de este artículo, no son de fácil generalización, debido a que se encuentra focalizado en el caso de dos caminos nodo disjuntos, para ajustar el criterio de dominancia entre caminos utilizado por SAMCRA, y el vector de restricciones C', utilizado al momento realizar búsquedas de caminos cuyo valor resulta ser $C' < 2C$, o sea menor al que utiliza DIMCRA. A pesar de ello, la estructura del algoritmo puede ser utilizada de la misma manera, para generar una versión exacta para el problema, aunque podría no resultar eficiente.

3.4. Heurística CADILAC

CADILAC, como ya se mencionó anteriormente, desarrolla una heurística basada en la metaheurística GRASP, que resuelve el problema de encontrar k caminos nodo disjuntos de largo acotado para un par de nodos terminales de una red. Este algoritmo tiene como estructura general seguir el esquema de esta metaheurística, utilizando un número máximo de iteraciones para detener la ejecución y acumulando la mejor solución en cada iteración. Respetando todo lo dicho anteriormente, existen variantes sobre el esquema general de GRASP, que se encuentran en los algoritmos de construcción de soluciones factibles y de búsqueda local.

Fase de construcción de CADILAC

La fase de construcción de soluciones factibles se basa en el algoritmo presentado por Bhandari, el cual se mencionó más arriba y se presentó su pseudocódigo. CADILAC toma la estructura básica de Bhandari y modifica algunos de sus elementos para de esa manera poder generar soluciones factibles aleatorias que cumplan con las restricciones del problema. Algunas de estas modificaciones se encuentran inspiradas en el algoritmo de DIMCRA [30]. Como se mencionó anteriormente, DIMCRA adapta el algoritmo de Bhandari para el caso en que los costos las aristas tienen más de una dimensión y el cálculo del largo de un camino no es lineal como es el caso de CADILAC.
En primer lugar se tiene la utilización de una función de costo no lineal en el cálculo. En el problema resuelto por CADILAC, el costo original y la cantidad de aristas de un camino se maneja como un costo de dos componentes. Por lo cual si se define una nueva función de costo $\vec{c} : \mathbb{E} \to \mathbb{R}^2$ donde $\vec{c}(e) = (c(e), 1), \forall e \in \mathbb{E}$.

El costo total de un camino P queda definido entonces como:

$$\vec{c}(P) = \{\sum_{e \in P} \vec{c}(e) \ si \sum_{e \in P} \vec{c}_1(e) \leq d, \infty \ en\, caso\, contrario\} \tag{3}$$

En segundo lugar se tiene la modificación de la restricción sobre el largo al hallar el camino más corto, que en el caso de CADILAC se controla por parámetro. En DIMCRA, en la fase de búsqueda del camino más corto, las restricciones se multiplican por 2, modificando al realizar la búsqueda de un camino la función $\vec{c}$ reemplazando d por $2d$. La idea de utilizar esta restricción y no la original, consiste en que de no hacerlo se reduciría el espacio de búsqueda, aumentando consecuentemente el riesgo de excluir soluciones factibles.

En la etapa posterior, se combina el camino encontrado con la solución parcial, si alguno de los caminos resultantes viola la restricción original, se descarta el último camino encontrado, menos las aristas superpuestas con la solución parcial, y se realiza la búsqueda de uno nuevo.

En CADILAC la restricción de la cantidad de aristas por camino, d, se relaja, cuando realiza la búsqueda del camino más corto y para ello utiliza un parámetro de entrada que multiplica a la restricción original. A este parámetro no se le dio ningún valor fijo ya que los autores de CADILAC no disponían de elementos teóricos suficientes para justificar un valor. Solamente pudo afirmarse que ese valor debe ser mayor que 1, debido a que un valor inferior restringiría el espacio de soluciones factibles y que cuanto más grande sea el valor, más lento será el algoritmo, debido a que causará que el algoritmo de búsqueda del camino más corto genere mayor cantidad de caminos que derivarán en soluciones no factibles.

En tercer lugar se toma del algoritmo de DIMCRA la idea de asignar costo (0,0), en lugar del valor opuesto como lo hace Bhandari, a las aristas que pertenecen a la solución parcial al momento de realizar la búsqueda de un nuevo camino.

Bhandari utiliza el valor opuesto porque, aunque el grafo resultante contenga aristas de costo negativo, no contendrá ciclos de costo negativo, lo que perjudicaría al algoritmo de búsqueda de caminos. Esto ocurre debido a que en cada paso, el algoritmo de Bhandari construye una solución que es minimal respecto a su costo total en el conjunto de soluciones factibles del problema considerando su misma cantidad de caminos. De manera concisa, en cada paso en el camino de hallar los k caminos, el algoritmo de Bhandari halla la solución para $1 \leq n < k$.

Por lo cual si al momento de invertir las aristas del grafo y asignarles su costo opuesto se genera un ciclo negativo, significará que dicha solución no era minimal, puesto que reemplazando el fragmento de la solución que pertenece a dicho ciclo, por el fragmento que no pertenece a la solución, se estaría hallando una solución parcial de menor costo lo que contradice la propiedad mencionada anteriormente.

La Figura 9 muestra un ejemplo de esto. Si en el algoritmo de Bhandari se halló el camino coloreado en negro entre s y t, y al invertir las aristas de dicho camino y asignar costo negativo a sus aristas se produjese un ciclo de costo negativo $C = a, b, c, d, e, a$, como en la figura, esto significa que el camino

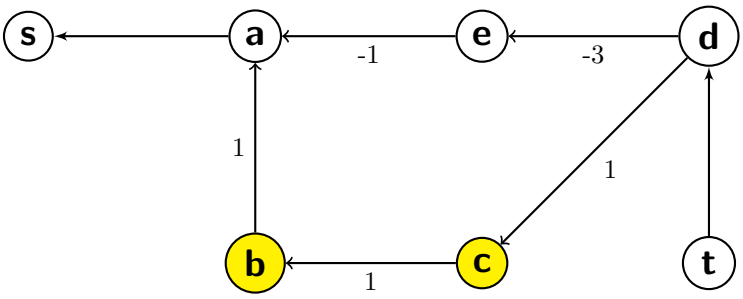

Figura 9: Ejemplo no resoluble por Bhandari.

hallado por el algoritmo de Dijkstra en la fase anterior no halló el camino más corto, debido a que si se sustituye el fragmento $P_1 = a, e, d$ por el fragmento $P_2 = a, b, c, d$, el resultado sería una solución de menor costo, lo cual es un absurdo.

En el caso de CADILAC, el algoritmo goloso aleatorio no genera soluciones óptimas sino aleatorias, como su nombre lo indica, utilizando un algoritmo para hallar caminos que no retorna el camino más corto. Es por esta razón que en el caso de CADILAC de ser posible se generan ciclos de costo negativo en el grafo al invertir las aristas y esto afectaría el funcionamiento del algoritmo. Por esta misma razón es que se le asigna un costo $\vec{c}(e) = (0, 0), \forall e \in Solucion$, al momento de invertir las aristas.

Además de los elementos incorporados de DIMCRA, CADILAC realiza otra modificación, la cual es la aleatorización del algoritmo de Dijkstra utilizado para hallar el camino más corto en cada paso del algoritmo, como se mencionó anteriormente. La aleatoriedad es sencilla y consiste en sortear con probabilidad p la posibilidad de visitar un vecino al estar procesando un nodo. Con esta modificación simple se puede obtener resultados aleatorios que no se alejan demasiado de la solución óptima, si el parámetro p se encuentra en un valor próximo a 1. El algoritmo de la Figura 10 contiene el pseudocódigo del algoritmo de Dijkstra Aleatorizado.

El algoritmo de la Figura 11 describe el algoritmo realizado por CADILAC para la generación de soluciones factibles para la fase de construcción de la heurística implementada en dicho algoritmo.

A diferencia del algoritmo de construcción de soluciones factibles de la metaheurística GRASP, no se genera una lista de candidatos implícita en la estructura del algoritmo, debido a que en cada paso, los elementos a seleccionar están limitados a los vecinos del nodo visitado.

Búsquedas locales de CADILAC

La fase de búsqueda local toma la solución factible generada en la fase de construcción por el algoritmo goloso aleatorio y aplica un algoritmo que recorre una estructura de vecindad en busca de una mejor solución.

Algoritmo 5 *DijkstraAleatorizado*(G, s, t, d, p)

```
1: // Camino asocia un camino entre s y t a cada nodo.
2: Camino[v] ← ∅, ∀v ∈ G;
3: // La cola de prioridad ordenada por costo
4: Cola ← Insertar(Cola, Camino[s]);
5: while not Vacia(Cola) do
6:    C ← ExtraerCaminoDeCostoMinimo(Cola);
7:    u ← Destino(C);
8:    if u = t then
9:       return C;
10:   else
11:      for each v ∈ Vecinos(G, u) do
12:         if Random(0, 1) < p then
13:            P ← C + (u, v)
14:            if Largo(P) ≤ d andCosto(Camino[v]) > Costo(P) then
15:               Eliminar(Cola, Camino[v]);
16:               Camino[v] ← P;
17:               Insertar(Cola, P);
18:            end if
19:         end if
20:      end for
21:   end if
22: end while
```

Figura 10: Pseudocodigo del algoritmo de Dijkstra aleatorizado.

Algoritmo 6 *AlgoritmoGolosoAleatorioCADILAC*$(G, C, k, MaxIntentos)$

1: $Solucion \leftarrow \emptyset$;
2: $Intentos \leftarrow MaxIntentos$;
3: **while** $k > 0$ **and**$Intentos > 0$ **do**
4: $G' \leftarrow Grafo()$;
5: $V(G') \leftarrow V(G)$;
6: $E(G') \leftarrow E(G) \backslash Solucion \cup \{(u, v) : (v, u) \in Solucion\}$;
7: $\vec{c} : E \rightarrow \mathbb{R}^+, \vec{c}(e) = \left\{ \begin{array}{l} \vec{c}(e)\ \forall e \notin Solucion, \\ (0, 0)\ \forall e \in Solucion \end{array} \right\}$;
8: $RP' \leftarrow CaminoAleatorio(Inicio, Fin, G', p)$;
9: $Solucion' \leftarrow (Solucion \cup SP') \setminus (Solucion \cap SP')$
10: **if** $EsFactible(Solucion')$ **then**
11: $Solucion \leftarrow Solucion'$;
12: $Intentos \leftarrow Maxintentos$;
13: $k \leftarrow k - 1$;
14: **else**
15: $Intentos \leftarrow Intentos - 1$;
16: **end if**
17: **end while**
18: **if** $k > 0$ **then**
19: **return** *Solucion*;
20: **else**
21: **return** Error('No fue posible hallar una solución');
22: **end if**

Figura 11: Pseudocodigo del algoritmo goloso aleatorio utilizado para generar soluciones factibles para la heurística.

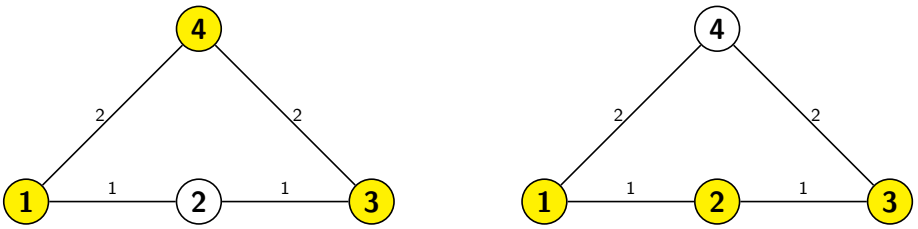

Figura 12: Se cambia el subcamino (1,4,3) por el (1,2,3), sustituyendo un nodo del camino por otro.

Estructura de Vecindad de CADILAC

La estructura de vecindad definida en CADILAC es la siguiente: $N : \mathbb{S} \times \mathbb{N}^2 \rightarrow 2^{\mathbb{S}}$, donde $\mathbb{S}$ es el conjunto de todas las soluciones factibles, transforma una solución $S \in \mathbb{S}$, dados dos parámetros n y m, en un conjunto de soluciones factibles que se denomina vecindad de S según N y se denota $N(S, n, m)$. El conjunto $N(S, n, m)$ es obtenido a partir de S mediante la aplicación de todos los movimientos del conjunto $M(S, n, m)$ que fueron definidos para el algoritmo CADILAC. El conjunto $M(S, n, m)$ está definido por todos los movimientos que reemplazan un subcamino de un camino de la solución S por otro subcamino, manteniendo la factibilidad de la solución resultante y con la condición de que el largo del camino a sustituir en S es menor o igual a n y que el del camino sustituto es menor o igual a m.

La Figura 12 muestra ejemplos de movimientos aplicados sobre un camino de una solución, donde se asume que dichos movimientos siempre mantienen la factibilidad de la solución final.

Una estructura de vecindad es transitiva si dadas dos soluciones existe una secuencia de movimientos dentro de esta que se transforman una solución en la otra. En el caso de CADILAC, la estructura de vecindad que fue definida para el algoritmo de búsqueda local no es transitiva, debido a que existen casos en los que no es posible transformar una solución en otra. La Figura 13 muestra un ejemplo de uno de estos casos. En esa figura se puede observar las soluciones para el problema de hallar dos caminos nodo-disjuntos en un mismo grafo. Fácilmente se puede ver que cualquier subcamino que intente reemplazar en alguno de los dos caminos siempre va a resultar en caminos superpuestos, dando como resultado una solución no factible, por lo que el vecindario de cada solución es vacío. Por lo cual, no es posible transformar una solución en otra realizando movimientos dentro de la estructura de vecindad, lo que demuestra que no es transitiva.

Algoritmo de Búsqueda en una Vecindad

El algoritmo de búsqueda define una estrategia por la cual se explora el espacio de soluciones definida por la estructura de vecindad N. Como fue mencionado anteriormente, el vecindario de una solución S, $N(S, n, m)$, se encuentra definido a partir de un conjunto de movimientos $M(S, n, m)$. Cada uno de los movimientos definidos por M es un cambio que se realiza sobre uno de los caminos de S, por lo que al momento de recorrer el vecindario de S resulta necesario

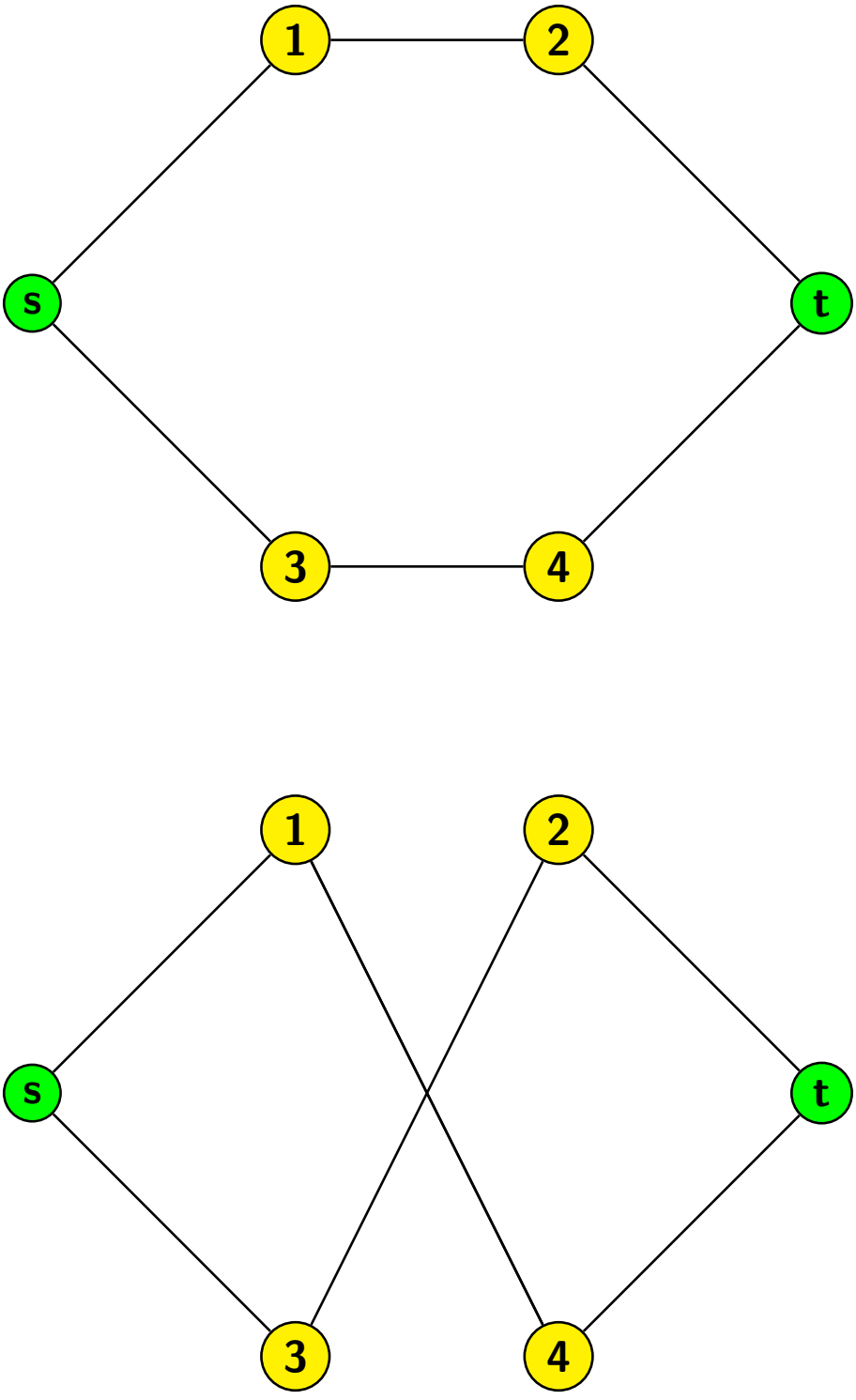

Figura 13: Dos soluciones para el problema de hallar dos caminos nodo-disjuntos en un mismo grafo.

Algoritmo 7 *EstructuraBusquedaLocal*(G, S, n, m)

```
1: while not CriterioFinalizacion() do
2:     P ← SeleccionarCamino(S);
3:     P' ← AplicarMovimiento(P, n, m);
4:     S ← (S \ P) ∪ P';
5: end while
6: return S;
```

Figura 14: Estructura general de un algoritmo de búsqueda local para la estructura de vecindad N.

definir sobre cuál camino $P \subseteq S$ se aplicará un movimiento. Luego de tener definido el camino P, se puede elegir cuál movimiento dentro de los posibles se va a aplicar. Así, de esta forma se obtiene una nueva solución S'. El proceso se puede repetir varias veces. El pseudocódigo presentado en la Figura 14 describe este algoritmo general para recorrer la estructura de vecindad.

El algoritmo de búsqueda local implementado en CADILAC respeta la estructura anteriormente definida, definiendo una forma de elegir un camino P de la solución S, una forma de elegir el movimiento que se aplica sobre P y el criterio de finalización de la búsqueda. El pseudocódigo definido en la Figura 15 describe el algoritmo de búsqueda local utilizado en CADILAC. En cada iteración, son recorridos todos los caminos de la solución S de forma aleatoria, sin repetirse ninguno. Para cada camino P, el mejor movimiento posible es seleccionado. El proceso se repite hasta que no hay más mejoras posibles, lo mismo que decir que se alcanza un óptimo local dentro de la estructura de vecindad.

El algoritmo en cuestión no corresponde a la categoría de *best-improving* o *first-improving* estrictamente, debido a que cada movimiento implica la selección aleatoria de uno de los caminos P, y luego en ese camino aplicar una estrategia *best-improving*. Por otro lado, si se considera la solución completa, dicho movimiento seleccionado no es necesariamente el mejor.

Algoritmo de Selección del Mejor Movimiento

El algoritmo que selecciona el movimiento a realizar sobre un camino elige el mejor posible según la estructura de vecindad N. Para lograrlo, son recorridos todos los nodos del camino y, para cada nodo, se toman los distintos subcaminos que comienzan en él, cuyo largo no sea mayor a n. Más adelante, para cada subcamino, se busca el camino más corto posible de largo menor o igual a m respetando la estructura de vecindad anteriormente definida. El pseudocódigo presentado en la Figura 16 muestra el proceso descrito.

Para el algoritmo de búsqueda del camino más corto, en una primera instancia se utilizó el algoritmo de Bellman-Ford debido a que este permite que se acote el largo del camino hallado. Por otro lado, al momento de realizar las pruebas preliminares del algoritmo, esto resultó ser lo suficientemente lento como para elevar demasiado el tiempo de ejecución de la heurística. Posterior-

Algoritmo 8 *BusquedaLocalCadilac*(G, k, s, n, m)

```
1: // Identificadores de los caminos de la solución S.
2: Posiciones ← [1..k];
3: repeat
4:    HuboMejoras ← false;
5:    Posiciones ← PermutarAleatoriamente(Posiciones);
6:    for each i ∈ Posiciones do
7:       P ← ObtenerCamino(S, i);
8:       P' ← AplicarMejorMovimiento(P, n, m);
9:       if Costo(P') < Costo(P) then
10:         S ← (S \ P) ∪ P';
11:         HuboMejoras ← true;
12:      end if
13:   end for
14: until not CriterioFinalizacion()
15: return S;
```

Figura 15: Algoritmo de búsqueda local utilizado para optimizar el resultado devuelto por la fase de construcción de la heurística CADILAC.

mente se probó con el algoritmo de Dijkstra, mejorando un poco los tiempos, aunque no lo suficiente. Por lo tanto, se diseñó una solución alternativa acorde al problema que se estaba resolviendo. De esta manera se llegó al algoritmo de búsqueda de caminos mediante el uso del algoritmo DFS (Depth First Search). Este algoritmo avanza a partir del origen y del destino en direcciones opuestas de forma recursiva, agregando aristas, en cada paso usando el esquema general de un algoritmo DFS. El algoritmo encuentra un camino cuando un camino parcial del origen se encuentra con un algoritmo parcial desde el destino. Además, no es mejor que Dijkstra o el de Bellman-Ford en el caso general. Pero para el caso de CADILAC, este algoritmo permite fácilmente limitar el largo máximo permitido para el camino hallado, y además limita cuantos niveles de recursión realiza el algoritmo. Tomando valores suficientemente pequeños para los tamaños máximos, la velocidad de ejecución de la heurística CADILAC es menor que con las opciones originales, y los resultados obtenidos por el algoritmo de búsqueda local no son significamente peores. El pseudocódigo es el presentado en la Figura 17.

Algoritmo 9 *AplicarMejorRendimiento*(P, c, d, n, m)

```
1: for each u ∈ P do
2:    s ← NodoOrigen(u);
3:    Costo ← 0;
4:    Largo ← 0;
5:    v ← u;
6:    // Para cada arista u del camino P.
7:    // Se recorren n aristas, incluyendo a u.
8:    repeat
9:       t ← NodoDestino(v);
10:      Costo ← Costo + c(v);
11:      Largo ← Largo + 1;
12:      CMax ← Costo + MejorIncr;
13:      LargoDisponible ← d − (Largo(P) − Largo);
14:      LMax ← min(LargoDisponible, m);
15:      R ← CaminoMasCorto(s, t, CMax, LMax);
16:      if Existe(R) then
17:         MejorReemplazo ← R;
18:         MejorIncr ← Costo(R) − Costo;
19:      end if
20:      v ← SiguienteArco(v, P);
21:   until Largo = n andExiste(v)
22: end for
23: if Existe(MejorReemplazo) then
24:    return Reemplazar(P, MejorReemplazo);
25: end if
```

Figura 16: Algoritmo que realiza el mejor movimiento posible dentro de la estructura de vecindad N sobre el camino P.

Algoritmo 10 *CaminoMascorto*$(OP, TP, CostoMax, LargoMax)$

```
1: // OP camino que sale de origen.
2: // TP camino que sale desde el destino.
3: u ← Destino(OP);
4: v ← Origen(TP);
5: Costo' ← Costo(OP) + Costo(TP);
6: Largo' ← Largo(OP) + Largo(TP);
7: if u = v and Costo' < Costo(SP) then
8:    SP ← OP + TP;
9: else
10:   if Largo' < LargoMax and ∃(u, v) ∈ E(G) and Costo' + Costo(u, v) <
      Costo(SP) then
11:      SP ← OP + (u, v) + TP;
12:   end if
13:   if Largo' + 2 ≤ LargoMax then
14:      for each (u, o) ∈ E(G) and (i, v) ∈ E(G) do
15:         CaminoMasCorto(OP + (u, o), (i, v) + TP);
16:      end for
17:   end if
18: end if
```

Figura 17: Algoritmo de búsqueda del camino más corto mediante el uso de un algoritmo DFS que inicia desde el nodo s y el nodo t.

4. Capítulo 4 - Resolución de 2NCON-BP-SN

4.1. Bloques de Metaheurística

Para poder construir una heurística y llegar a una solución factible del 2NCON-BP-SN se utilizará como base uno de los algoritmos presentado en el Doctorado de Robledo [14].

En dicha tesis Robledo aborda el problema de diseño de una red WAN descomponiéndolo en dos subproblemas inter-relacionados los cuales son: El diseño de la Red de Acceso (The Access Network Design Problem - ANDP) y el Diseño de la Red Dorsal (The Backbone Network Design Problem - BNDP).

Con el fin de entender el algoritmo a utilizar, se debe tener en claro ciertas definiciones previas las cuales se detallan a continuación:

Definición 24. *S_D es el conjunto total de nodos de la red.*

Definición 25. *$S_D^{(I)}$ es un conjunto de nodos distinguidos de la red, los cuales se denominan terminales.*

Definición 26. *Los nodos que se encuentren en $S_D \setminus S_D^{(I)}$ son denominados Nodos de Steiner.*

Definición 27. *$C = c_{i,j\,i,j \in S_D}$ es una matriz que contiene los costos de las aristas y en el caso de no existir una arista entre dicho par de nodos se tiene $c_{i,j} = \infty$.*

Definición 28. *$R = \{r_{i,j}\}_{i,j \in S_D^{(I)}}$ es una matriz que contiene la cantidad de caminos nodo disjuntos entre dos nodos incluidos en $S_D^{(I)}$ que en este caso en particular, siempre será 2.*

Definición 29. *$E = \{(i,j); \forall i,j \in S_D$ tal que $c_{i,j} < \infty\}$ es el conjunto de conexiones factibles entre los nodos pertenecientes a S_D.*

Definición 30. *$G = (S_D, E)$ es un grafo no dirigido que modela las conexiones de la red.*

Definición 31. *Γ es el espacio de soluciones factibles asociado con el problema.*

Definición 32. ***key-node:*** *Dada una instancia y una solución factible al problema $g_{sol} \in \Gamma$ se define un **key-node** como un nodo no terminal (perteneciente a $S_D \setminus S_D^{(I)}$) con grado por lo menos tres en g_{sol}.*

Definición 33. ***key-path:*** *Dada una instancia y una solución factible al problema $g_{sol} \in \Gamma$, se define **key-path** a un camino en g_{sol} donde todos sus nodos intermedios son no terminales con grado dos en g_{sol} y sus extremos son terminales o key-nodes.*

Definición 34. *Dada una solución factible $g_{sol} \in \Gamma$. Si se cumple que cada arista de g_{sol} pertenece a un camino entre dos nodos terminales, entonces es posible descomponer g_{sol} en key-paths, o lo que es lo mismo, existe un conjunto de key-paths para los cuales, toda arista de g_{sol} pertenece a uno y solo un key-path. Se denotará $K(g_{sol}) = (p_1, \ldots, p_h)$ a la descomposición de g_{sol} en key-paths ordenada por costos de mayor a menor.*

Definición 35. ***key-tree:*** *Dada una instancia y una solución factible al problema $g_{sol} \in \Gamma$ y además un key-node $v \in g_{sol}$, se define un* ***key-tree*** *asociado con v como el árbol en g_{sol} que se encuentra formado por todos los key-paths que tienen a v como uno de sus extremos.*

4.1.1. Fase de construcción

Una vez entendidas estas definiciones, se pasará a explicar el algoritmo a utilizar. Básicamente se utilizará el algoritmo **ConstPhase** de Takahashi-Matsuyama ya modificado por Robledo en su doctorado, utilizando CADILAC, el cual ya fue explicado anteriormente y permite hallar dados dos nodos, dos caminos nodo disjuntos.

Tanto para la utilización de CADILAC, como del algoritmo que a continuación se planteará, se debe tener la precaución que no tienen en cuenta, que los Nodos de Steiner puedan tener algún costo adicional al ser utilizados en nuestra solución, por lo cual en todo el GRASP se debe tener en cuenta dichos costos adicionales.

Para resolver el problema que surge con los costos de construcción de los nodos de Steiner se utilizará un GRASP con memoria.

Inicialmente, a todas las aristas que utilicen algún nodo de Steiner, se le sumará el costo de construcción del mismo al costo de la arista. De esta manera, de ser utilizada, se tendrá en cuenta el costo de construcción por utilizar un nodo de Steiner.

Cada vez que un nodo de Steiner es agregado a la solución en construcción, se retirará el costo de construcción, agregado a los costos de las aristas que lo utilizan a excepción de en una de las aristas que ya está en la solución y que tiene como origen o destino ese nodo de Steiner.

Para el momento de las búsquedas locales, se debe recordar cuál arista tiene almacenado en su costo, el costo de construcción del nodo de Steiner que utiliza, ya que si se retirara la misma, se tendrá que pasar dicho costo a otra arista que utilice ese nodo de Steiner si es que existiera.

Este algoritmo intenta construir una red de forma iterativa cumpliendo los requerimientos dados por la matriz $R = \{r_{i,j}\}_{i,j \in S_D^{(I)}}$ de requerimientos la cual, para este problema, es siempre 2 camino nodo disjuntos entre todo par de nodos terminales.

Dicho algoritmo utiliza las siguientes estructuras, las cuales deben estar debidamente inicializadas de la siguiente forma:

- La solución actual g_{sol} con los nodos de $S_D^{(I)}$ y un conjunto vacío de aristas.
- La matriz $M = \{m_{i,j}\}_{i,j \in S_D^{(I)}}$ donde $m_{i,j} = r_{i,j}, \forall i,j \in S_D^{(I)}$ la cual indica los requerimientos que aún no se han cumplido en la solución actual.
- El conjunto $P = \{P_{i,j}\}_{i,j} \in S_D^{(I)}$ con $P_{i,j} = \emptyset, \forall i,j \in S_D^{(I)}$ utilizado para guardar los $r_{i,j}$ caminos computados entre dos nodos terminales $i,j \in S_D^{(I)}$.
- La matriz $A = \{A_{i,j}\}_{i,j \in S_D^{(I)}}$ con $A_{i,j} = 0, \forall i,j \in S_D^{(I)}$ que será utilizada para guardar los lugares donde no ha sido encontrado un camino entre dos nodos terminales.

Algoritmo 11 *FaseDeConstruccion*(G_B, C, R, k)

1: $g_{sol} \leftarrow (S_D^{(I)}, \emptyset); m_{ij} \leftarrow r_{ij} \forall i, j \in S_D^{(I)}; P_{i,j} \leftarrow \emptyset \ \forall i, j \in S_D^{(I)}; A_{ij} \leftarrow 0 \ \forall i, j \in S_D^{(I)}$;
2: **while** $\exists m_{ij}$ tal que $A_{ij} <$ MAXINTENTOS **do**
3: Sea $i, j \in S_D^{(I)}$ un par de terminales elegidos randómicamente tal que $m_{ij} > 0$;
4: $\hat{g} \leftarrow (G_B \backslash P_{ij})$;
5: Sea $\hat{g}$ una matriz dada por $\hat{c_{uv}} \leftarrow \left\{ \begin{array}{l} 0 \textbf{ if } (u,v) \in g_{sol}, \\ c_{uv} \textbf{ if } (u,v) \in \hat{g} \backslash g_{sol} \end{array} \right\}$;
6: $l_p \leftarrow$ Los dos caminos nodos disjuntos entre i y j, considerando la matriz $\hat{c}$;
7: **if** $l_p = \emptyset$ **then**
8: $A_{ij} \leftarrow A_{ij} + 1$;
9: **else**
10: **if** $\exists \hat{p} \in l_p$ tal que $COST_{\hat{c}}(\hat{p}) = 0$ **then**
11: $p \leftarrow \hat{p}$;
12: **else**
13: $p \leftarrow SelectRandom(l_p); g_{sol} \leftarrow g_{sol} \cup \{p\}$;
14: $p_{ij} \leftarrow p_{ij} \cup \{p\}; m_{ij} \leftarrow m_{ij} - 1$;
15: $[P, M] \leftarrow ActualizarMatriz(g_{sol}, P, M, p, i, j)$;
16: **end if**
17: **end if**
18: **end while**
19: **return** g_{sol}, P

Figura 18: Pseudocodigo de Fase de Construcción.

En la Figura 18 se puede observar el algoritmo utilizado en la fase de construcción del GRASP. En dicho algoritmo se puede observar en la línea 1 la inicialización que se mencionó anteriormente.

En la línea 2 se comienza el bucle en el cual se va a buscar una solución factible al problema. El mismo será ejecutado hasta que se hayan cubierto todos los requerimientos pedidos (en este caso serían todos los pares de caminos nodo disjuntos entre todo par de nodos terminales) o ya se haya llegado a determinada cantidad de intentos para todos los pares de nodos terminales a los que falte hallar el par de caminos nodo disjuntos o al menos uno de los caminos.

En la línea 3 se escoge un par de nodos terminales randómicamente para los cuales aún falte hallar el par de caminos nodo disjuntos o al menos uno de los mismos.

Luego en la línea 4 se obtiene la red formada por la red original exceptuando cualquier arista o nodo perteneciente al camino que une el nodo i con el nodo j.

En la línea 5 se utiliza la red obtenida en la línea anterior para así al buscar él o los caminos faltantes entre i y j se tiene que el costo de las aristas serán:

- 0 si la arista ya se está utilizando en g_{sol}, la cual es la solución que se está construyendo.
- c_{uv} si la arista se encuentra en la red hallada en la línea anterior, exceptuando las que se encuentran en g_{sol}

Luego en la línea 6 se almacena en l_p a los dos caminos nodo disjuntos entre los nodos i y j hallados utilizando CADILAC.

En el bloque entre las líneas 7 y 17 se analiza los caminos encontrados de la siguiente forma:

1. Si no se logra encontrar ningún camino entre i y j. Es decir que l_p es el conjunto vacío se actualiza la cantidad de intentos realizados en la búsqueda de los caminos nodo disjuntos para el par de nodos terminales i y j. Finalmente se regresa a la línea 2 del algoritmo para continuar la búsqueda de la solución factible.
2. Si l_p es distinto del conjunto de vacío se tiene dos casos:
   a) Si existe un camino $\hat{p}$ perteneciente a l_p que tenga costo cero entonces dicho camino es guardado.
   b) En caso contrario, se selecciona un camino de los pertenecientes a l_p y se agrega a g_{sol}. También se guarda en p_{ij} ese camino, como uno de los caminos nodo disjuntos que unen estos dos nodos terminales, y finalmente se disminuye la cantidad de caminos faltantes entre i y j. En este caso también se invoca otro algoritmo, denominado **ActualizarMatriz**, que será explicado a continuación y lo que éste realiza, es ir actualizando otros valores y estructuras que son necesarias para que esta fase de construcción sea exitosa.

Una vez terminado el bloque que encierra las líneas 2 a la 18 simplemente se retorna g_{sol} la cual en caso de haber encontrado una solución se encuentra almacenada en esa variable. Además, se devuelve P que es el conjunto de caminos nodo disjuntos entre todo par de nodos terminales.

El algoritmo **ActualizarMatriz** que se utiliza en la fase de construcción es el presentado en la Figura 19, y es el que se encarga de verificar si de manera no intencional al hallar un camino entre los nodos terminales i y j, se encontró por accidente un camino entre i y k y entre k y j siendo k también un nodo terminal.

Desde la línea 1 a la 14 se puede observar el bucle que realiza este algoritmo. Para cada nodo pertenecientes a los nodos terminales tales que sean distinto del nodo i y j, y que además pertenezcan al camino p se realiza lo siguiente:

1. Se verifica si la cantidad de caminos entre los nodos i y k es mayor que cero en la línea 2 del algoritmo. Si esto es verdadero, se verifica que la intersección de los caminos entre el conjunto de Caminos, los nodos i y k, y el camino encontrado entre i y k, tiene solamente como elementos los nodos i y k.

   De esta manera, se ha encontrado otro camino nodo disjunto entre los nodos i y k y se puede agregar al conjunto $P_{i,k}$ así como también restar en 1 la cantidad de caminos nodo disjuntos pedidos de este par de nodos.

Algoritmo 12 *ActualizarMatriz*(g_{sol}, P, M, p, i, j)

```
1: for each k ∈ S_D^(I), k ≠ i, j tal que k ∈ p do
2:   if m_ik > 0 then
3:     if (Nodos(P_ik) ∩ Nodos(p_(i,k)) = {i, k}) then
4:       P_ik ← P_ik ∪ {P_(i,k)}
5:       m_ik ← m_ik − 1; m_ki ← m_ki − 1;
6:     end if
7:   end if
8:   if m_kj > 0 then
9:     if (Nodos(P_kj) ∩ Nodos(p_(k,j)) = {k, j}) then
10:      P_kj ← P_kj ∪ {P_(k,j)}
11:      m_kj ← m_kj − 1; m_jk ← m_jk − 1;
12:    end if
13:  end if
14: end for
15: return P, M
```

Figura 19: Pseudocódigo de ActualizarMatriz.

2. Luego se realiza el mismo mecanismo, pero ahora pensando en los nodos k y j. Se verifica si la cantidad de caminos entre los nodos k y j es mayor que cero en la línea 2 del algoritmo. Si esto es verdadero se verifica que la intersección de los caminos entre el conjunto de Caminos, los nodos k y j, y el camino encontrado entre j y k, tiene solamente como elementos los nodos k y j.

   De esta manera, se ha encontrado otro camino nodo disjunto entre los nodos k y j y se puede agregar al conjunto $P_{k,j}$ así como también restar en 1 la cantidad de caminos nodo disjuntos pedidos de este par de nodos.

4.1.2. Intercambio de "Key tree"

En esta búsqueda local se intentará, a partir de la mejor solución encontrada hasta el momento, encontrar un key-tree y sustituirlo por otro de menor costo.

Las hojas de un key-tree pueden llegar a ser nodos terminales, por lo cual estos nodos no podrán ser retirados de la solución.

Para la construcción de caminos alternativos de los que utilizaban aristas del key-tree encontrado, y que ya no pertenecerán a la solución, es que se volverá a utilizar de manera similar a la fase de construcción, el algoritmo CADILAC.

Sea $G = (V, E)$ el grafo original y $T \subseteq V$ el conjunto de nodos denominados terminales. Por otro lado, se tiene el conjunto $\hat{N}$ el cual contiene los nodos de Steiner que se encuentran por fuera de la solución encontrada hasta el momento, de aquí en adelante denominada G_{sol}. A su vez se tiene:

- τ_v el cual es un $key - tree$ dentro de G_{sol}.

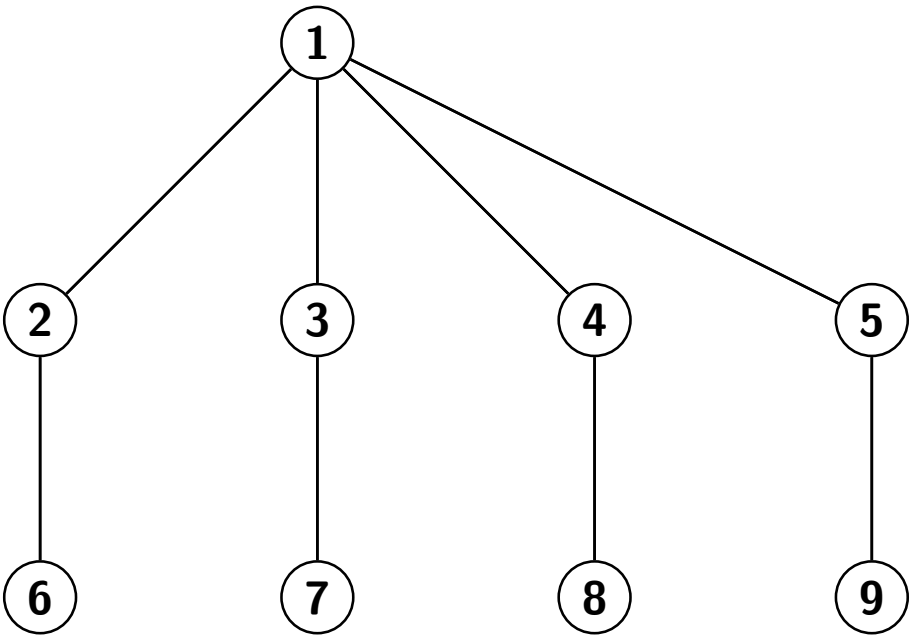

Figura 20: Key-Tree.

- $\hat{S}$ es el conjunto de *nodosdeSteiner* de τ_v que no son hoja.

- $\hat{V}$ son las hojas de τ_v.

En la Figura 20 se puede observar un ejemplo de Key-tree (τ_v) en donde la raíz sería el nodo 1 y de él "cuelgan" 4 key-paths de los cuales sus hojas son los nodos 6, 7, 8 y 9.

$\hat{S}$ es el conjunto de nodos 1, 2, 3, 4 y 5, los cuales son los nodos de Steiner, que no son hoja.

Por último, se tiene $\hat{V}$ que serían los nodos 6, 7, 8 y 9.

Una vez encontrado un key-tree se agrega un nodo z el cual se encuentra conectado con costo cero a todos los nodos pertenecientes al $key - tree$ que son raíces del mismo. De esta manera se tiene el grafo de la Figura 21.

Además, sea $G(\hat{V} \cup \hat{N} \cup \hat{S}) \cup \{(z, i) | i \in \hat{V}\} = \hat{H}$.

Con todas estas nuevas definiciones claras se aplica lo siguiente:

$\forall s \in \hat{S} \cup \hat{N}$ aplico $CADILAC$ desde S hacia z con $k = |\hat{V}|$ y $d =$ Mínimo de los largos de los $key - paths$ de τ_v.

Si se tiene éxito se encontrará un nuevo $key - tree$ el cual tendrá un costo menor al inicial y por la definición de $key - tree$, la solución encontrada seguirá siendo factible.

El pseudocódigo correspondiente a dicha búsqueda local se encuentra en la Figura 22.

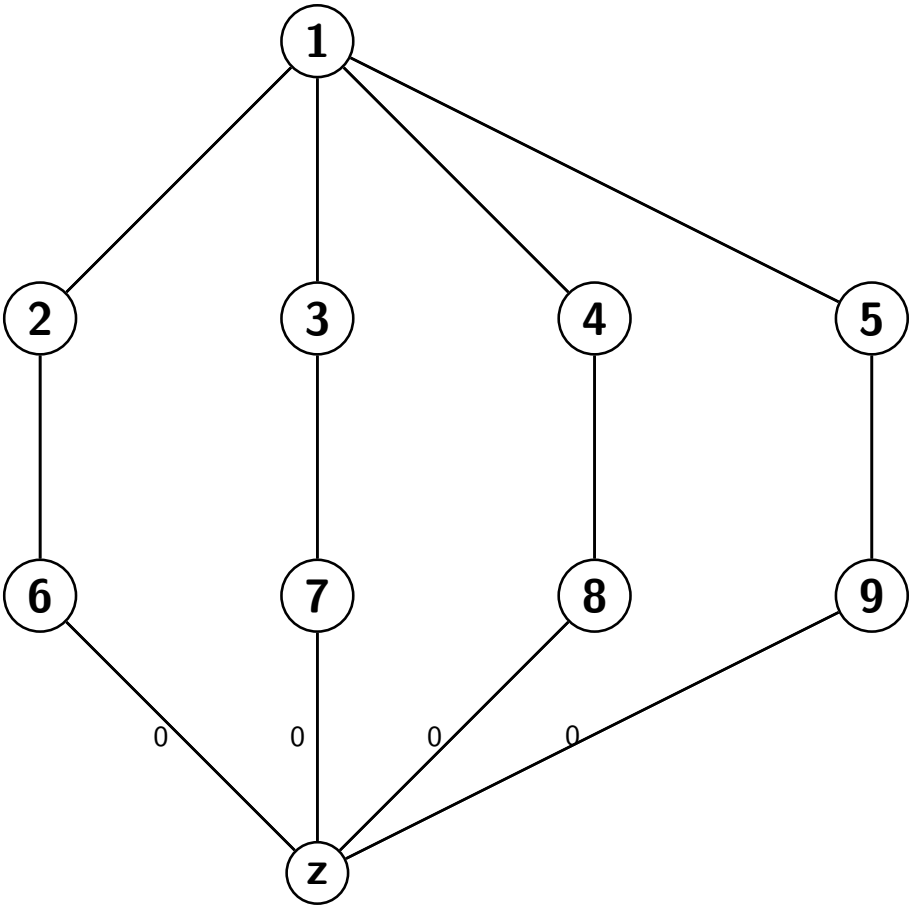

Figura 21: Key-Tree modificado.

Algoritmo 13 *BusquedaLocal1*$(g_{orig}, g_{sol}, \hat{S}, \hat{V}, \tau_v, \hat{H}, costoSolActual)$

1: **for each** $s \in \hat{S} \cup \hat{N}$ **do**
2: $solObtenida \leftarrow$ Aplico CADILAC, $k = |\hat{V}|$ y $d =$ Mínimo de los largos de los $Key - Paths$ de τ_v
3: **if** $costoSolObtenida < costoSolActual$ **then**
4: $costoSolActual = solObtenida$
5: Sustituyo el $Key - Tree$ en cuestión por la solución encontrada exceptuando sus hojas.
6: Se actualiza g_{sol}
7: $SalirForEach$
8: **end if**
9: **end for**
10: **return** g_{sol}

Figura 22: Pseudocódigo de Búsqueda Local 1.

En dicho pseudocódigo se puede ver que para todo elemento perteneciente al conjunto $\hat{S} \cup \hat{N}$ (línea 1) se aplica CADILAC desde S hacia z con $k = |\hat{V}|$ y $d =$ Mínimo de los largos de los $key - paths$ de τ_v (línea 2).

En la línea inmediatamente posterior (línea 3) se compara el costo de esta nueva solución encontrada con el costo de la mejor solución encontrada hasta el momento.

De ser menor el costo de la solución que se acaba de encontrar, en las líneas 4, 5 y 6 se procede a la actualización del nuevo mejor costo de la red, sustituir el Key-Tree por la nueva solución encontrada exceptuando sus raíces y se actualiza g_{sol}.

Luego de dichas actualizaciones se procede a salir del **for each**. Si la comparación de costos de la línea 3 no fuese exitosa entonces se continúa la recorrida en el bucle hasta finalizar el mismo o encontrar una solución mejor a la hallada hasta ese momento.

4.1.3. Minimizar "Nodos de Steiner"

En esta búsqueda local se buscará en la solución los Nodos de Steiner que no sean imprescindibles para la misma.

Si se tienen dos nodos A y B respectivamente, ambos comunicados con un tercer nodo C no terminal (Nodo de Steiner) y este último nodo es de grado dos en la solución, se intentará prescindir del mismo para así tener una solución final de menor costo.

Para poder quitar el nodo C y continuar teniendo una solución factible es necesario chequear que en la ausencia de dicho nodo se sigue teniendo entre todo par de nodos terminales dos caminos nodo disjuntos con un largo menor al establecido en los requerimientos para cada par de terminales.

Por otro lado se chequea la posibilidad de que si existe en el grafo original la arista directa entre los nodos A y B y no fue tenida en cuenta para la solución

Algoritmo 14 *BusquedaLocal2*(g_{orig}, g_{sol})

1: **for each** $n \in g_{sol}$ **do**
2: **if** $gr(n) == 2$ *and not* $esTerminal(n)$ **then**
3: **if** $\forall (x, y) \in g_{sol} - n$ *and* $esTerminal(x)$ *and* $esTerminal(y)$ *and* $\exists k$ caminos nodo disjuntos entre (x,y), siendo $k \geq 2$ **then**
4: Retirar nodo n de la solución encontrada hasta el momento y posiblemente tener que agregar aristas u otros nodos para que nuestra solución continúe siendo factible.
5: Se actualiza g_{sol}
6: **end if**
7: **end if**
8: **end for**
9: **return** g_{sol}

Figura 23: Pseudocódigo de Búsqueda Local 2.

encontrada en la fase de construcción, se intentará añadir dicha arista que une los vértices A y B, retirar el nodo C y verificar que la solución continúa siendo factible.

El pseudocódigo correspondiente a dicha búsqueda local se encuentra en la Figura 23.

Dicho algoritmo es una recorrida por todos los nodos de g_{sol}, la cual se puede ver en el bloque **for each** que comprende las líneas 1 a la 8.

En dicho bucle solamente interesan los nodos que tengan grado igual a 2 y además no sean terminal. Ambas condiciones sobre los nodos se verifican en la línea 2 del algoritmo.

Luego en la linea 3 del algoritmo se examina las condiciones descritas anteriormente para las cuales ese nodo estaría innecesariamente en la solución. Esa línea es la encargada de verificar si al retirar dicho nodo y en algunos casos agregando aristas y/o otro nodo a la solución continúa siendo factible.

Finalmente en la línea 4 se retira el nodo n de la solución encontrada hasta el momento, y se busca la forma de hacer que la solución siga siendo factible agregando otros nodo y/o aristas que sean necesarias para sustituir el nodo que se retira.

Si luego de todos estos cambios se llega a una solución de menor costo que la que se tenía al arrancar con esta búsqueda local, se actualiza g_{sol} como es mencionado en la línea 5.

Un ejemplo de esta búsqueda local es el presentado en la Figura 24. En la cual los nodos de color verde son los terminales y los rojos nodos de Steiner.

En esa red se puede ver que existen nodos de Steiner que son de grado 2 y al no ser terminales no serían estrictamente necesarios en la red final a menos que la inserción de los mismos disminuyan el costo total de la solución.

Para el caso de la red del ejemplo, los nodos de Steiner 9, 10 y 11 serían innecesarios ya que no aportan a la solución final. Estos nodos son los que esta búsqueda local intenta eliminar de la solución.

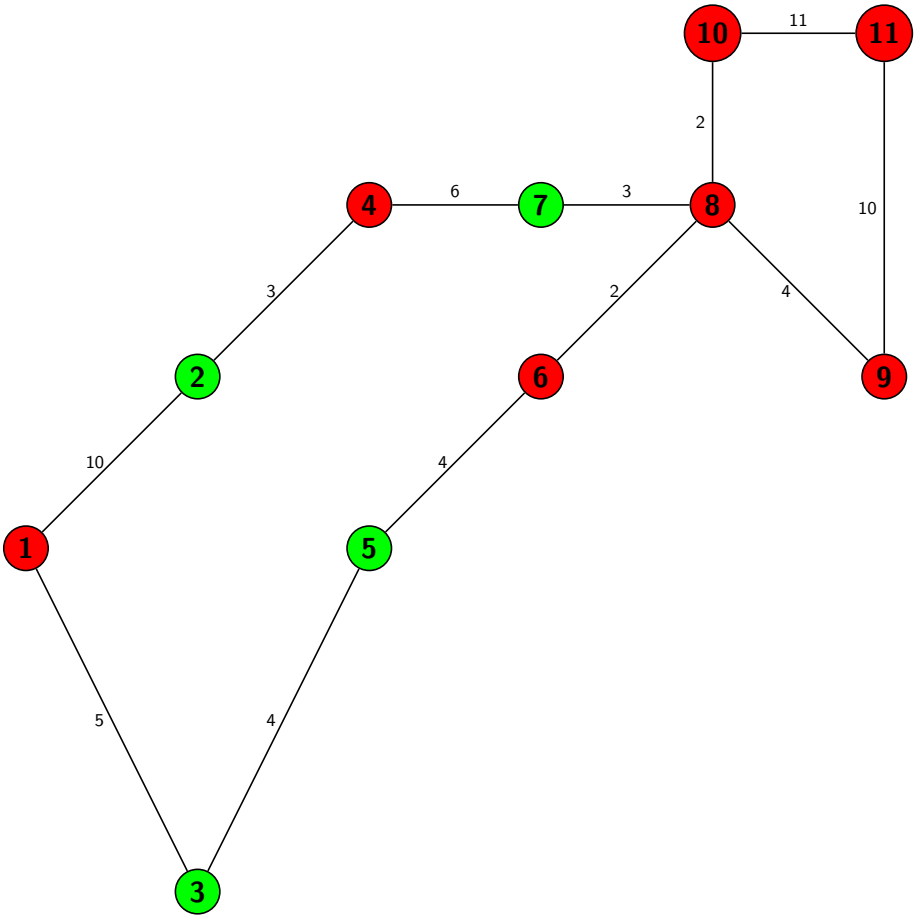

Figura 24: Grafo de entrada para búsqueda local 2.

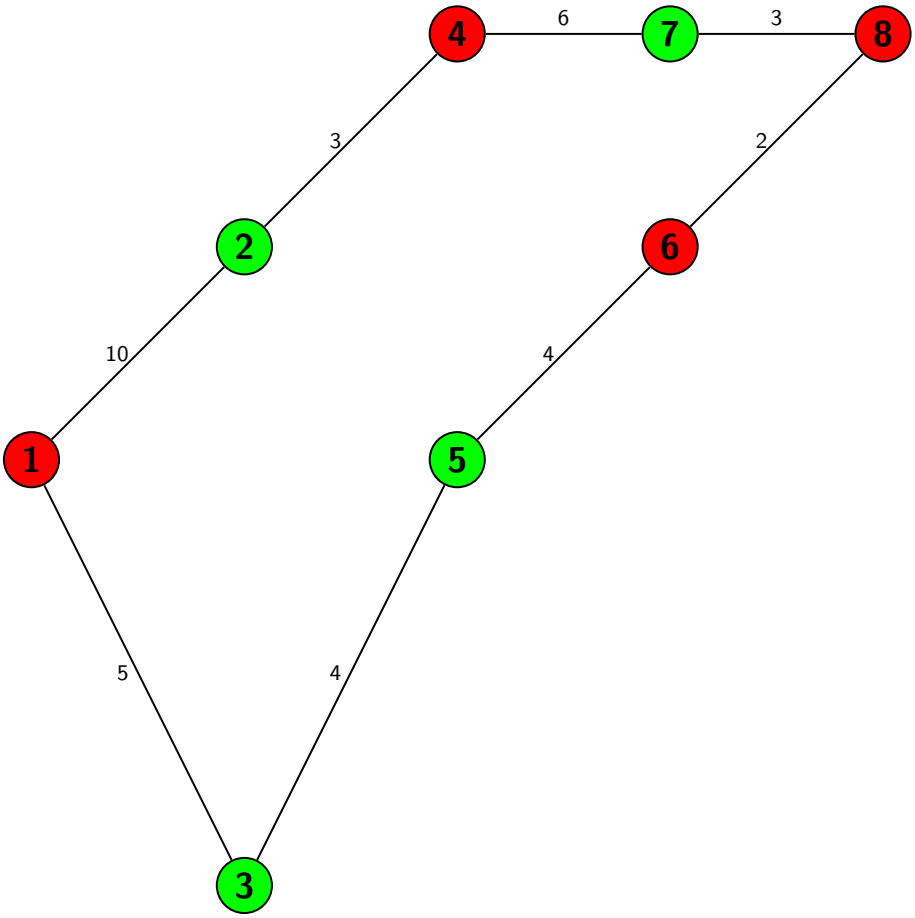

Figura 25: Grafo salida de la búsqueda local 2.

La red resultante luego de aplicar la búsqueda local 2 quedaría como en la Figura 25. Por lo cual luego de aplicar la misma, se disminuiría el costo final de la red.

4.1.4. Eliminando aristas muy caras

Esta búsqueda local tiene como objetivo intentar librarse de las aristas de mayor costo que se tengan en la mejor solución encontrada hasta el momento. Para realizar esto se utilizará un teorema mencionado anteriormente que permitirá sustituir un conjunto de aristas por otro y seguir manteniendo la factibilidad de la solución.

Siguiendo el teorema 4 definido previamente en la sección de teoremas relevantes del área, sea G la mejor solución encontrada hasta el momento de iniciar esta búsqueda local.

La mejor solución encontrada hasta el momento cumple las hipótesis de ser 2 nodo conexa, por lo tanto se puede utilizar como G, para aplicar el teorema de Monmma.

Algoritmo 15 *BusquedaLocal3*(g_{orig}, g_{sol})

1: $G \leftarrow g_{sol}$
2: $G' \leftarrow$ Subgrafo inducido por un subconjunto de vértices de g_{sol}
3: $G'' \leftarrow$ Aplico Cadilac, para hallar otra posible solución más barata que la que teníamos hasta el momento.
4: $G^* \leftarrow$ Aplicamos el teorema 4 mencionado en la sección de fundamentos.
5: **if** *costoSolObtenida* $<$ *costoSolActual* **then**
6: *costoSolActual* = *solObtenida*;
7: Se actualiza g_{sol};
8: **end if**
9: **return** g_{sol};

Figura 26: Pseudocódigo de Búsqueda Local 3.

Además se toma como G', al subgrafo inducido por un subconjunto de vértices, donde los mismos poseen gran parte de las aristas de mayor costo. De esta manera, si se logra sustituir el conjunto de aristas de este subgrafo inducido, se logrará eliminar de la solución varias aristas que tenían un costo grande y así lograr minimizar el costo total de la solución.

Luego mediante la ayuda de CADILAC se hallará G'' el cual será 2-conexo y tendrá como conjunto de nodos el mismo que tenía G'. De esta manera se hallará G^* y si su costo total es menor que el costo encontrado hasta el momento se tendrá una mejor solución por la aplicación del teorema 4.

El pseudocódigo de esta búsqueda local es el presentado en la Figura 26.

En la línea 1 del mismo se asigna a G la mejor solución obtenida hasta el momento ya que como se mencionó anteriormente cumple con las hipótesis del teorema al ser 2 nodo conexa. De esta misma manera se asigna a G' un subgrafo inducido por un subconjunto de vértices tal cual como fue descrito anteriormente de manera que la mayoría de las aristas de mayor costo que se tienen en la solución se encuentren en este subgrafo.

Luego, aplicando Cadilac se hallará G'' que es quien estaría faltando para cumplir todas las hipótesis necesarias para poder aplicar el teorema 4 en la línea 4 del algoritmo.

Finalmente en las líneas posteriores, si se encontró una solución de mejor costo que la que se tenía al comienzo de esta búsqueda local, se actualizará la mejor solución encontrada hasta el momento. Luego en la línea 9 se retorna la mejor solución encontrada o la que se tenía antes de comenzar la búsqueda local.

4.1.5. Eliminando nodos con grado muy alto

Esta búsqueda local, al igual que la anterior, utiliza Monmma pero otro de sus teoremas. En este caso se utiliza el teorema 2 mencionado en la sección de fundamentos. Utilizando este teorema es posible hallar otra solución factible de menor costo ya que a excepción de cumplir con la restricción de los largos de los caminos, se sabe que existe otra solución con menor grado de su vértices y

Algoritmo 16 *BusquedaLocal4*(g_{orig}, g_{sol})

```
1: if ∃ subgrafo G' de g_sol con algún nodo con grado > 3 then
2:    G'' ← Aplico Cadilac, para hallar otra posible solución más barata
3:    if costoSolObtenida(G'') < costoSolAnterior(G') then
4:       Se actualiza g_sol;
5:    end if
6: end if
7: return g_sol
```

Figura 27: Pseudocódigo de Búsqueda Local 4.

posiblemente más barata.

Siguiendo el Teorema 2 enunciado anteriormente, se tiene que si en la mejor solución hallada hasta el momento, se tiene un subgrafo, el cual es 2-conexo y alguno de ellos son de grado > 3, entonces es posible que exista una solución con un costo menor que siga siendo dos nodo conexa y además respete la cantidad de hops que se tiene fija como requerimientos.

Por lo tanto, basándose en este teorema se tiene que si se continúa respetando la restricción de largo de los caminos, es posible mejorar el costo total de la solución. Dicha construcción de ese nuevo subgrafo se realizará mediante CADILAC, respetando las restricciones de largo del problema.

El pseudocódigo de esta búsqueda local es el presentado en la Figura 27.

En la línea 1 de dicho pseudocódigo se verificará si existe un subgrafo de la mejor solución encontrada hasta el momento que cumpla las hipótesis del teorema mencionado. Si existe en la línea siguiente se aplicará CADILAC en la búsqueda de una mejor solución a la encontrada hasta el momento, ya que por lo dicho anteriormente existe la posibilidad de que exista y que sea más barata. En la línea 3 se comparan los costos entre la solución encontrada y la que se tenía anteriormente. De ser menor se actualiza g_{sol}. Finalmente en la línea 7 se retorna g_{sol} modificado o igual a como estaba antes de la búsqueda local si no se encontró mejora.

4.1.6. Memoria en el GRASP

En esta búsqueda local se intentará explotar toda la información obtenida en el GRASP, como ser cuáles nodos de Steiner aún se tienen en la mejor solución encontrada hasta el momento. También se posee información sobre qué caminos entre dos nodos terminales utilizan dichos nodos de Steiner, y de esta manera se podrá saber cuales nodos de Steiner son los menos utilizados. Teniendo esta información se pasará a retirar el nodo de Steiner que se utiliza menos en la solución. Por esta razón la solución pasará a no cumplir la 2-nodo-conectividad entre determinados nodos terminales. Por lo cual es necesario hallar nuevos caminos para lo cual se utilizará CADILAC forzando a que no se utilice el nodo de Steiner que se acaba de retirar. Si se encuentra una mejor solución se volverá a repetir este mecanismo hasta que no se logre mejorar el costo.

Algoritmo 17 *BusquedaLocal5*(g_{orig}, g_{sol})

1: $n \leftarrow n$ es el nodo de Steiner que se utiliza menos en nuestra solución.
2: $G' \leftarrow g_{sol}$ menos el nodo n.
3: $G'' \leftarrow$ Aplico Cadilac, para hallar otra posible solución más barata y sin el nodo de Steiner n.
4: **if** $costoSolObtenida(G'') < costoSolAnterior(G_{sol})$ **then**
5: Se actualiza g_{sol}
6: **end if**
7: **return** g_{sol}

Figura 28: Pseudocódigo de Búsqueda Local 5.

El pseudocódigo de esta búsqueda local es el presentado en las Figura 28.

En la línea 1 del algoritmo se busca cuál es el nodo de Steiner que es menos utilizado dentro de la mejor solución encontrada hasta el momento g_{sol}. Este nodo es el retirado de g_{sol} en la línea 2 para así no tenerlo en cuenta en la búsqueda de mejores soluciones.

En la tercer línea, nuevamente aplicando CADILAC y sin tener en cuenta el nodo n, se intenta buscar una nueva y mejor solución a g_{sol}.

En la línea 4 se consulta sobre el costo de esta nueva solución comparado contra el que ya se tenía con g_{sol}, y de tener éxito en la búsqueda de una solución más barata, se procede a la actualización de g_{sol} en la línea 5.

Finalmente en la línea 7 se retorna g_{sol} ya modificado de haber sido exitosa la búsqueda local.

4.1.7. Diagrama de flujo y pseudocódigo del GRASP

El diagrama de flujo presentado en la Figura 29 representa el algoritmo completo desarrollado para resolver el problema 2NCON-BP-SN. En dicho diagrama se puede ver la fase de construcción y en qué orden se ejecutaron las distintas búsquedas locales del GRASP utilizado.

El pseudocódigo completo de dicho GRASP se encuentra en la Figura 30. En la línea 1 del mismo se invoca a la fase de construcción del GRASP. Esta etapa es la que se describió anteriormente e irá llamando a CADILAC en varias oportunidades para ir encontrando entre un par de nodos terminales dos caminos nodo disjuntos de menor costo y que no superen los d hops que tiene como requirmiento este problema. Una vez terminada esta etapa, se debe analizar si la red devuelta por la fase de construcción es factible. De no ser así, se retornará que no existe solución. Este chequeo se realiza en las líneas 2, 3 y 4 del algoritmo de la Figura 30.

Luego en las sucesivas líneas se llamarán a las búsquedas locales anteriormente definidas y explicadas en este Capítulo. Dichas búsquedas locales retornarán la misma red recibida como entrada si no fue posible encontrar una mejor solución, o de lo contrario retornarán la mejor solución encontrada hasta el momento.

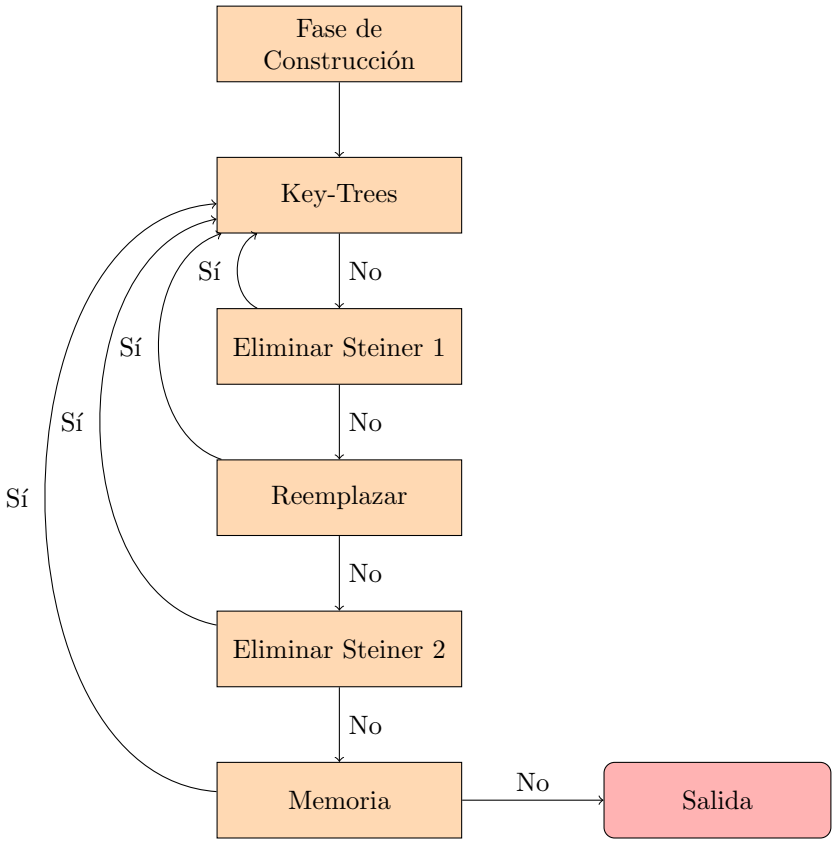

Figura 29: Diagrama de flujo del GRASP utilizado para el 2NCON-BP-SN.

Una vez terminada la ejecución de las 5 búsquedas locales y que las mismas no proporcionen ninguna mejora a la mejor solución encontrada se retorna G_{sol} (línea 25 del algoritmo).

Algoritmo 18 *GRASP − Completo*(g_{orig}, g_{sol})

1: $G_{sol} = FasedeConstruccion(G_{orig})$;
2: **if** $NoEsSolucionFactible(G_{sol})$ **then**
3: **return** "No existe solucion";
4: **end if**
5: $G_{sol} = BusquedaLocal1(G_{sol})$;
6: **if** $Mejora(G_{sol})$ **then**
7: **GOTO** 5;
8: **end if**
9: $G_{sol} = BusquedaLocal2(G_{sol})$;
10: **if** $Mejora(G_{sol})$ **then**
11: **GOTO** 5;
12: **end if**
13: $G_{sol} = BusquedaLocal3(G_{sol})$;
14: **if** $Mejora(G_{sol})$ **then**
15: **GOTO** 5;
16: **end if**
17: $G_{sol} = BusquedaLocal4(G_{sol})$;
18: **if** $Mejora(G_{sol})$ **then**
19: **GOTO** 5;
20: **end if**
21: $G_{sol} = BusquedaLocal5(G_{sol})$;
22: **if** $Mejora(G_{sol})$ **then**
23: **GOTO** 5;
24: **end if**
25: **return** g_{sol}

Figura 30: Pseudocódigo GRASP - Completo.

Algoritmo 19 *SolucionGreedy*$(G_{Orig}, L_{Ord}, G_{Sol})$

1: **while** *not* $esSolFactible(G_{Sol})$ **do**
2: Sea (i, j) la arista más barata, la remuevo de L_{Ord} y la inserto en G_{Sol}.
3: **end while**
4: **return** g_{sol}

Figura 31: Solución ávida o greedy.

5. Capítulo 5 - Resultados

En este capítulo se presentarán los distintos resultados obtenidos en este proyecto, así como también la manera de obtenerlos, ya sea selección de qué grafos utilizar, cantidad de nodos terminales, cantidad de Nodos de Steiner y limitante de hops.

Como ya se dijo anteriormente, no se encontraron trabajos específicos en el área que resolvieran el problema aquí planteado, por lo que se necesitó la implementación de una solución ávida o greedy con el fin de poder comparar resultados. Dicha solución, se encuentra explicada y acompañada de su pseudocódigo, en la subsección siguiente.

Por otro lado, también fue necesario obtener un variado juego de datos para realizar las pruebas. Esto se logró utilizando la librería TSPLIB, la cual será explicada en este capítulo.

Por último se presentarán varias tablas con los resultados obtenidos, a partir de la metaheurística aquí desarrollada y a partir de la solución greedy construída a efectos comparativos.

5.1. Solución ávida o greedy

La solución contra la cual se decidió contrastar los resultados obtenidos por la metaheurística aquí desarrollada es muy simple.

El algoritmo desarrollado en esta solución, parte de una lista de aristas ordenadas (L_{Ord}) de menor a mayor de la red original según su costo.

El procedimiento realizado por el mismo, es ir agregando de una arista por vez, partiendo de la lista antes mencionada, hasta llegar a una solución que cumpla todos los requerimientos pedidos por el problema planteado para este proyecto o concluir que no existe solución al problema a partir de la red recibida como entrada.

Esta solución deberá, en cada iteración, controlar si no se llegó a una solución factible del problema. Esto ya de por sí es muy costoso, debido a que teniendo una solución, el poder decir si la misma es factible o no, es un problema **NP-Completo**.

La entrada de este algoritmo, así como también de la metaheurística aquí desarrollada, son grafos brindados por la librería TSPLIB la cual será explicada en la próxima sección.

El pseudocódigo del algoritmo en cuestión es el presentado en la Figura 31.

5.2. TSPLIB

La librería TSPLIB es conocida por poseer ejemplos de instancias que resuelven el problema **TSP** (y problemas relacionados) provenientes de diversos orígenes y de diferentes tipos que se detallan a continuación.

- **Symmetric traveling salesman problem (TSP)**
- **Hamiltonian cycle problem (HCP)**
- **Asymmetric traveling salesman problem (ATSP)**
- **Sequential ordering problem (SOP)**
- **Capacitated vehicle routing problem (CVRP)**

5.3. Ejecución de metaheurística y resultados

Para la ejecución de la metaheurística aquí desarrollada y la solcuión greedy se tomaron instancias asociadas al traveling salesman problem (TSP). En la selección de las mismas, se tuvo en cuenta que fuesen de variada cantidad de nodos para así poder examinar resultados de la metaheurísticas con diferentes cantidad de nodos. Los nombres de los grafos elegidos se encuentran en la primer columna de las tablas que figuran a continuación y la cantidad de nodos totales de dicha red, es el número que acompaña su nombre.

A cada instancia extraída de la $TSPLIB$ se la utilizó nueve veces pero de diferente manera. En las primeras tres ejecuciones se tomaron el 30 % de los nodos como terminales, en las siguientes tres 40 % y finalmente en las últimas tres el 50 %. Para cada una de esas ejecuciones también fue necesario calcular la cantidad de hops máximo que puede tener cada camino del par nodo disjunto que une cada par de terminales. Dicha limitante de hops tomó tres valores diferentes, uno con un valor fijo en 5, con el fin de evaluar el comportamiento del algoritmo para una limitante muy pequeña. Los restantes valores no son fijos, en una de las corridas es el 30 % con respecto al porcentaje de los terminales elegidos en dicha ejecución, y la última corrida corresponde al 15 % con respecto al mismo número.

Para todas las pruebas relizadas se contó con el acceso al cluster de la Facultad de Ingeniería [39] para, de esta manera, poder realizar pruebas con mayor rapidez y de mayor cantidad de nodos.

Cuadro 1: Resultados

				GRASP/VND		Greedy		
Grafo	**Terminales**	**Hops**	**Steiner**	**Costo**	**Tiempo (s)**	**Costo**	**Tiempo (s)**	**GAP**
a280.tsp	84	5	196	1448,35	7536	-	-	-
a280.tsp	84	12	196	1254,52	7225	-	-	-
a280.tsp	84	25	196	1200,32	7295	-	-	-
a280.tsp	112	5	168	1500,41	8564	-	-	-
a280.tsp	112	16	168	1845,12	8778	-	-	-
a280.tsp	112	33	168	1965,96	8796	-	-	-
a280.tsp	140	5	140	-	-	-	-	-
a280.tsp	140	21	140	-	-	-	-	-
a280.tsp	140	42	140	-	-	-		-
bier127.tsp	38	5	89	896,32	6598	-	-	-
bier127.tsp	38	6	89	842,25	6641	-	-	-
bier127.tsp	38	11	89	810,45	6789	-	-	-
bier127.tsp	50	5	77	799,47	6788	-	-	-
bier127.tsp	50	7	77	815,68	6693	-	-	-
bier127.tsp	50	15	77	844,17	7145	-	-	-
bier127.tsp	63	5	64	816,25	7342	-	-	-
bier127.tsp	63	10	64	800,91	7894	-	-	-
bier127.tsp	63	19	64	899,23	8841	-	-	-
brd14051.tsp	4215	5	9836	-	-	-	-	-
brd14051.tsp	4215	632	9836	-	-	-	-	-
brd14051.tsp	4215	1264	9836	-	-	-	-	-
brd14051.tsp	5620	5	8431	-	-	-	-	-
brd14051.tsp	5620	843	8431	-	-	-	-	-
brd14051.tsp	5620	1686	8431	-	-	-	-	-
brd14051.tsp	7025	5	7026	-	-	-	-	-
brd14051.tsp	7025	1053	7026	-	-	-	-	-
brd14051.tsp	7025	2107	7026	-	-	-	-	-
d15112.tsp	4533	5	10579	-	-	-	-	-
d15112.tsp	4533	680	10579	-	-	-	-	-
d15112.tsp	4533	1360	10579	-	-	-	-	-
d15112.tsp	6044	5	9068	-	-	-	-	-
d15112.tsp	6044	906	9068	-	-	-	-	-
d15112.tsp	6044	1813	9068	-	-	-	-	-
d15112.tsp	7556	5	7556	-	-	-	-	-
d15112.tsp	7556	1133	7556	-	-	-	-	-
d15112.tsp	7556	2266	7556	-	-	-	-	-
Eli51.tsp	15	2	36	512,23	250	621,66	260	-17,60
Eli51.tsp	15	4	36	524,45	222	634,45	269	-17,33
Eli51.tsp	15	5	36	527,78	267	656,74	287	-19,63
Eli51.tsp	20	3	31	523,16	289	526,65	356	-0,66
Eli51.tsp	20	5	31	578,61	365	554,23	384	-4,39
Eli51.tsp	20	6	31	585,32	342	565,34	399	-3,53
Eli51.tsp	25	3	26	521,39	401,21	545,32	451	-4,38
Eli51.tsp	25	5	26	546,74	375,41	556,41	474	-1,73
Eli51.tsp	25	8	26	549,96	379,65	561,24	499	-2,01

Cuadro 2: Continuación Resultados

				GRASP/VND		Greedy		
Grafo	**Terminales**	**Hops**	**Steiner**	**Costo**	**Tiempo (s)**	**Costo**	**Tiempo (s)**	**GAP**
Eil76.tsp	22	3	54	125,46	124	156,98	149	-20,07
Eil76.tsp	22	5	54	147,54	180	159,47	196	-8,08
Eil76.tsp	22	6	54	152,32	201	174,65	200	-14,65
Eil76.tsp	30	4	46	174,64	208	189,66	256	-8,60
Eil76.tsp	30	5	46	125,41	215	196,32	285	-56,54
Eil76.tsp	30	9	46	196,24	218	199,31	288	-1,56
Eil76.tsp	38	5	38	225,14	211	204,39	298	-9,21
Eil76.tsp	38	6	38	224,51	225	265,45	315	-18,23
Eil76.tsp	38	11	38	229,34	224	279,77	327	-18,02
Eil101.tsp	30	4	71	199,64	200	310,51	334	-55,53
Eil101.tsp	30	5	71	211,79	145	314,4	356	-48,44
Eil101.tsp	30	9	71	234,66	217	368,45	388	-57,01
Eil101.tsp	40	5	61	255,98	215	377,87	401	-47,61
Eil101.tsp	40	6	61	284,11	249	389,14	403	-36,96
Eil101.tsp	40	12	61	266,49	224	396,41	415	-48,75
Eil101.tsp	50	5	51	301,41	286	401,54	436	-33,22
Eil101.tsp	50	8	51	302,49	299	402,49	461	-33,05
Eil101.tsp	50	15	51	328,58	311	403,64	477	-22,84
Fnl4461.tsp	1338	5	3123	-	-	-	-	-
Fnl4461.tsp	1338	201	3123	-	-	-	-	-
Fnl4461.tsp	1338	402	3123	-	-	-	-	-
Fnl4461.tsp	1784	5	2677	-	-	-	-	-
Fnl4461.tsp	1784	267	2677	-	-	-	-	-
Fnl4461.tsp	1784	535	2677	-	-	-	-	-
Fnl4461.tsp	2230	5	2231	-	-	-	-	-
Fnl4461.tsp	2230	334	2231	-	-	-	-	-
Fnl4461.tsp	2230	669	2231	-	-	-	-	-
nrw1379.tsp	413	5	966	-	-	-	-	-
nrw1379.tsp	413	62	966	-	-	-	-	-
nrw1379.tsp	413	123	966	-	-	-	-	-
nrw1379.tsp	551	5	828	-	-	-	-	-
nrw1379.tsp	551	82	828	-	-	-	-	-
nrw1379.tsp	551	165	828	-	-	-	-	-
nrw1379.tsp	689	5	690	-	-	-	-	-
nrw1379.tsp	689	103	690	-	-	-	-	-
nrw1379.tsp	689	206	690	-	-	-	-	-
kroA150.tsp	45	7	105	300,22	319	-	-	-
kroA150.tsp	45	14	105	312,45	285	-		-
kroA150.tsp	60	5	90	320,78	356	-	-	-
kroA150.tsp	60	9	90	315,44	486	-	-	-
kroA150.tsp	60	18	90	322,19	451	-	-	-
kroA150.tsp	75	5	75	304,32	399	-	-	-
kroA150.tsp	75	11	75	328,56	415	-	-	-
kroA150.tsp	75	22	75	333,88	418	-	-	-

Cuadro 3: Continuación Resultados

				GRASP/VND		Greedy		
Grafo	**Terminales**	**Hops**	**Steiner**	**Costo**	**Tiempo (s)**	**Costo**	**Tiempo (s)**	**GAP**
pr1002.tsp	300	5	702	-	-	-	-	-
pr1002.tsp	300	45	702	-	-	-	-	-
pr1002.tsp	300	90	702	-	-	-	-	-
pr1002.tsp	400	5	602	-	-	-	-	-
pr1002.tsp	400	60	602	-	-	-	-	-
pr1002.tsp	400	120	602	-	-	-	-	-
pr1002.tsp	501	5	501	-	-	-	-	-
pr1002.tsp	501	125	501	-	-	-	-	-
pr1002.tsp	501	250	501	-	-	-	-	-
rat575.tsp	172	5	403	-	-	-	-	-
rat575.tsp	172	26	403	-	-	-	-	-
rat575.tsp	172	51	403	-	-	-	-	-
rat575.tsp	230	5	345	-	-	-	-	-
rat575.tsp	230	34	345	-	-	-	-	-
rat575.tsp	230	69	345	-	-	-	-	-
rat575.tsp	287	5	288	-	-	-	-	-
rat575.tsp	287	43	288	-	-	-	-	-
rat575.tsp	287	86	288	-	-	-	-	-
rat783.tsp	234	5	549	-	-	-	-	-
rat783.tsp	234	35	549	-	-	-	-	-
rat783.tsp	234	70	549	-	-	-	-	-
rat783.tsp	313	5	470	-	-	-	-	-
rat783.tsp	313	47	470	-	-	-	-	-
rat783.tsp	313	93	470	-	-	-	-	-
rat783.tsp	392	5	391	-	-	-	-	-
rat783.tsp	392	58	391	-	-	-	-	-
rat783.tsp	392	117	391	-	-	-	-	-
kroB200.tsp	60	5	140	504,34	515	-	-	-
kroB200.tsp	60	8	140	514,22	524	-	-	-
kroB200.tsp	60	18	140	599,39	568	-	-	-
kroB200.tsp	80	5	120	623,33	522	-	-	-
kroB200.tsp	80	12	120	649,69	525	-	-	-
kroB200.tsp	80	24	120	655,85	539	-	-	-
kroB200.tsp	100	5	100	601,88	504	-	-	-
kroB200.tsp	100	16	100	641,25	509	-	-	-
kroB200.tsp	100	30	100	625,45	518	-	-	-
kroC100.tsp	30	4	70	322,36	365	-	-	-
kroC100.tsp	30	5	70	369,48	398	-	-	-
kroC100.tsp	30	9	70	368,96	605	-	-	-
kroC100.tsp	40	5	60	388,47	451	-	-	-
kroC100.tsp	40	6	60	396,48	496	-	-	-
kroC100.tsp	40	12	60	385,55	541	-	-	-
kroC100.tsp	50	5	50	389,65	569	-	-	-
kroC100.tsp	50	8	50	396,41	594	-	-	-
kroC100.tsp	50	15	50	401,81	597	-	-	-

6. Capítulo 6 - Conclusiones

En este proyecto se abordó el problema de encontrar una red dos nodo conexa de mínimo costo con nodos de Steiner y caminos acotados (2NCON-BP-SN). Se realizó una búsqueda bibliográfica y no se ha encontrado publicaciones que ataquen este problema con este enfoque.

En dicha búsqueda bibliográfica, se encontró CADILAC, que es el trabajo de unos compañeros de hace unos años, y resuelve un sub problema del resuelto en este proyecto.

CADILAC [1] permitió de manera rápida, para un par de nodos terminales, hallar un par de caminos nodo disjuntos, y mediante la ejecución de varias veces dicho algoritmo llegar a una solución preliminar del problema. Dicha solución fue mejorada mediante 5 búsquedas locales distintas, desarrolladas específicamente para este problema.

El presente trabajo fue basado en la metodología GRASP, la cual fue debidamente explicada en la sección 1.3.2 en el Capítulo 1. Luego de evaluar las distintas alternativas para abordar este problema, se eligió GRASP debido al nivel de complejidad que contiene el mismo.

Se necesitó la implementación de una solución alternativa para poder efectuar una comparación. Observando los resultados del capítulo anterior, para gran parte de los grafos utilizados, se obtuvo una solución al problema. La mayoría de los grafos que no se pudieron encontrar soluciones es debido a su tamaño (cantidad de nodos) y que se debió cortar la ejecución del programa para continuar con la siguiente prueba luego de muchas horas de ejecución. También se comprobó que la única limitante que se encontró, fue la cantidad de nodos de la red que se recibía como entrada y los casos que pusimos una limitante de hops muy chica. En todos los casos la metaheurística obtuvo mejores resultados que la solución greedy.

En muchos casos como se puede apreciar en el Capítulo anterior la solución Greedy no logró encontrar una solución y se tuvo que cortar en la mayoría de los casos por tiempo. Algunos casos de la metaheurística ocurrió esto mismo pero en menor cantidad.

También para los casos que fue posible se calculó el GAP correspondiente y en todos los casos dio como resultado que la metaheurística superó a la solución Greedy.

Para poder probar grafos de tamaño medianos y grandes, se solicitó a la Facultad de Ingeniería acceso a su Cluster para así tener un mejor tiempo de cómputo y esto se notó en el tiempo final de las pruebas realizadas, aunque para grafos muy grandes no se logró obtener resultados.

7. Capítulo 7 - Trabajo Futuro

Como posible trabajo futuro sería de utilidad hacer que el programa sea más tenga mayor nivel de parametrización.

Por ejemplo: Durante la realización de las pruebas se notó que sería de utilidad si se pudiese especificar qué nodos en particular se desea sean los nodos terminales (distinguidos), y no que internamente el programa lo decida, como sucedió para las distintas pruebas que se realizaron.

También pensando en la usabilidad del software desarrollado, sería desable que los costos entre los nodos no fuese basado solamente entre las distancias de los mismos, sino que se pueda establecer como entrada el costo entre los nodos.

Todas estas mejoras harían que el programa desarrollado se pueda usar de manera más sencilla. Además, hasta el momento siempre se trabaja sobre grafos completos, lo cual reduce mucho la cantidad de redes que se puede recibir como entrada.

Existen muchas mejoras que se pueden realizar a la metaheurística obtenida, como las mencionadas anteriormente, que por falta de tiempo no se hizo. Esto ocurrió, debido a que durante el proyecto, se dedicó gran parte del tiempo a investigar, encontrar trabajos del área de este problema o similares, que pudiese ayudar a la resolución del mismo y entender a fondo el problema a tratar.

Referencias

[1] L. Chiappara, "Caminos nodo-disjuntos, acotados y de menor costo entre dos nodos," https://rev-inv-ope.univ-paris1.fr/spip.php?article63, Facultad de Ingeniería, Montevideo, Uruguay, Thesis, 2014.

[2] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[3] M. Stoer, *Design of survivable networks*. Springer-Verlag Berlin, 1992, vol. 1531.

[4] F. Harary, "Graph theory," 1969.

[5] G. Fritz, "Heuristic methods for the hop constrained survivable network design problem," Technische Universität Wien, Fakultät für Informatik der Technischen Universität Wien, Master Thesis, 2011.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company, 1979.

[7] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *j-PROC-LONDON-MATH-SOC-2*, vol. 42, pp. 230–265, 1936, this is the paper that introduced what is now called the *Universal Turing Machine*.

[8] C. C. Ribeiro and P. Hansen, *Essays and surveys in Metaheuristics*, ser. Operations Research/Computer Science Interfaces Series. Boston, Dordrecht, London: Kluwer academic publishers, 2002. [Online]. Available: http://opac.inria.fr/record=b1107635

[9] M. Resende and C. Ribeiro, "Greedy randomized adaptive search procedures," in *Handbook of metaheuristics*. Springer New York, 2003, pp. 219–249.

[10] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[11] J. Brimberg, P. Hansen, N. Mladenović, and E. D. Taillard, "Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem," *Operations Research*, vol. 48, no. 3, pp. 444–460, 2000.

[12] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[13] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[14] F. Robledo, "Grasp heuristics for wide area network design," Ph.D. dissertation, Universidad de la República, 2005.

[15] H. Kerivin and A. R. Mahjoub, "Design of survivable networks: A survey," *Networks*, vol. 46, no. 1, pp. 1–21, 2005.

[16] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[17] C. L. Monma, B. S. Munson, and W. R. Pulleyblank, "Minimum-weight two-connected spanning networks," *Mathematical Programming*, vol. 46, no. 1-3, pp. 153–171, 1990.

[18] M. Labbé, G. Laporte, I. R. Martín, and J. J. S. González, "The median cycle problem," 1999.

[19] S. Kedad-Sidhoum and V. H. Nguyen, "An exact algorithm for solving the ring star problem," *Optimization*, vol. 59, no. 1, pp. 125–140, 2010.

[20] M. Labbé, G. Laporte, I. R. Martín, and J. J. S. González, "The ring star problem: Polyhedral analysis and exact algorithm," *Networks*, vol. 43, no. 3, pp. 177–189, 2004.

[21] R. Baldacci, M. Dell'Amico, and J. S. González, "The capacitated m-ring-star problem," *Operations Research*, vol. 55, no. 6, pp. 1147–1162, 2007.

[22] A. Hill and S. Voß, "Optimal capacitated ring trees," *EURO Journal on Computational Optimization*, vol. 4, no. 2, pp. 137–166, 2016.

[23] M. Salari, Z. Naji Azimi, and P. Toth, "An integer linear programming based heuristic approach for the capacitated m-ring-star problem," in *4rd international conference of Iranian Operations Research Society*, 2011.

[24] H. I. Calvete, C. Galé, and J. A. Iranzo, "An efficient evolutionary algorithm for the ring star problem," *European Journal of Operational Research*, vol. 231, no. 1, pp. 22–33, 2013.

[25] G. Bayá, A. Mauttone, F. Robledo, and P. Romero, "Capacitated m two-node survivable star problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 253–260, 2016.

[26] R. Recoba, F. Robledo, P. Romero, and O. Viera, "Two-node-connected star problem," *International Transactions in Operational Research*, Early view (to appear), 2016.

[27] R. Bhandari, *Survivable networks: algorithms for diverse routing.* Springer Science & Business Media, 1999.

[28] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[29] J. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.

[30] Y. Guo, F. Kuipers, and P. Van Mieghem, "Link-disjoint paths for reliable qos routing," *International Journal of Communication Systems*, vol. 16, no. 9, pp. 779–798, 2003.

[31] A. Itai, Y. Perl, and Y. Shiloach, "The complexity of finding maximum disjoint paths with length constraints," *Networks*, vol. 12, no. 3, pp. 277–286, 1982. [Online]. Available: http://dx.doi.org/10.1002/net.3230120306

[32] A. Beshir and F. Kuipers, *Variants of the min-sum link-disjoint paths problem.* IEEE/SCVT, 2011.

[33] K. Xiong, Z.-d. Qiu, Y. Guo, and H. Zhang, "Multi-constrained shortest disjoint paths for reliable qos routing," *ETRI journal*, vol. 31, no. 5, pp. 534–544, 2009.

[34] Y. Kobayashi and C. Sommer, "On shortest disjoint paths in planar graphs," *Discrete Optimization*, vol. 7, no. 4, pp. 234–245, 2010.

[35] R. Fleischer, Q. Ge, J. Li, and H. Zhu, "Efficient algorithms for k-disjoint paths problems on dags," in *International Conference on Algorithmic Applications in Management.* Springer, 2007, pp. 134–143.

[36] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974. [Online]. Available: http://dx.doi.org/10.1002/net.3230040204

[37] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths." *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[38] P. Van Mieghem, H. De Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Computer Networks*, vol. 37, no. 3, pp. 407–423, 2001.

[39] "Cluster-Fing," https://www.fing.edu.uy/cluster/index.php, 2008.

[40] T. L. Magnanti and S. Raghavan, "Strong formulations for network design problems with connectivity requirements," *Networks*, vol. 45, no. 2, pp. 61–79, 2005.

[41] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71. New York, NY, USA: ACM, 1971, pp. 151–158. [Online]. Available: http://doi.acm.org/10.1145/800157.805047

[42] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem. A correction," *j-PROC-LONDON-MATH-SOC-2*, vol. 43, pp. 544–546, 1937, see [7]. [Online]. Available: http://turing.ecs.soton.ac.uk/browse.php/B/12

[43] S. Hawking, *God created the integers: the mathematical breakthroughs that changed history.* Philadelphia, PA and London, UK: Running Press Book Publishers, 2005.

[44] M. G. Resendel and C. C. Ribeiro, "Grasp with path-relinking: Recent advances and applications," in *Metaheuristics: progress as real problem solvers.* Springer US, 2005, pp. 29–63.

[45] G. Zorpette, "Keeping the phone lines open," *j-IEEE-SPECTRUM*, pp. 32–36, 1989.

[46] V. Rutenburg, "Efficient distributed algorithms for computing shortest pairs of maximally disjoint paths in communication networks," in *Proceedings IEEE INFOCOM '91, The Conference on Computer Communications, Tenth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking in the 90s, Bal Harbour, Florida, USA, April 7-11, 1991*, 1991, pp. 1214–1219.

[47] A. Srinivas and E. Modiano, "Finding minimum energy disjoint paths in wireless ad-hoc networks," *Wireless Networks*, vol. 11, no. 4, pp. 401–417, 2005. [Online]. Available: http://dx.doi.org/10.1007/s11276-005-1765-0

[48] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing (The Springer International Series in Engineering and Computer Science)*. Springer, 1998.

[49] Y. Guo, F. A. Kuipers, and P. V. Mieghem, "Link-disjoint paths for reliable qos routing," *Int. J. Communication Systems*, vol. 16, no. 9, pp. 779–798, 2003. [Online]. Available: http://dx.doi.org/10.1002/dac.612

[50] A. Itai, Y. Perl, and Y. Shiloach, "The complexity of finding maximum disjoint paths with length constraints," *Networks*, vol. 12, no. 3, pp. 277–286, 1982. [Online]. Available: http://dx.doi.org/10.1002/net.3230120306

[51] E. Köhler, R. H. Möhring, and H. Schilling, "Fast point-to-point shortest path computations with arc-flags," in *IN: 9TH DIMACS IMPLEMENTATION CHALLENGE [29*, 2006.

[52] R. Bhandari, *Survivable Networks: Algorithms for Diverse Routing (The Springer International Series in Engineering and Computer Science)*. Springer, 1998.

[53] M. G. Resende-Pardalos, *Handbook of Optimization in Telecommunications*. Springer, 2006.

[54] H. Takahashi and A. Matsuyama, "An approximate solution for the steiner problem in graphs," *Math. Japonica*, vol. 24, no. 6, pp. 573–577, 1980.

[55] F. Robledo, "Diseño topológico de redes, casos de estudio: The generalized steiner problem, y the steiner 2-edge-connected subgraph problem," Ph.D. dissertation, Master Thesis. InCo, PEDECIBA Inform tica, 2000.

[56] F. Harary, "The maximum connectivity of a graph," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 48, no. 7, p. 1142, 1962.

[57] P. Winter, "Steiner problem in networks: a survey," *Networks*, vol. 17, no. 2, pp. 129–167, 1987.

[58] F. Robledo and E. C. Bentacourt, "Designing backbone networks using the generalized steiner problem," in *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on.* IEEE, 2009, pp. 327–334.

[59] F. Robledo, H. Cancela, and G. Rubino, "Solving the steiner two-node-survivable network problem," *International Journal of Logistics Systems and Management*, vol. 6, no. 2, pp. 218–234, 2010.

[60] S. Soni, R. Gupta, and H. Pirkul, "Survivable network design: The state of the art," *Information Systems Frontiers*, vol. 1, no. 3, pp. 303–315, 1999.

[61] S. Soni and H. Pirkul, "Design of survivable networks with connectivity requirements," *Telecommunication Systems*, vol. 20, no. 1-2, pp. 133–149, 2002.

[62] Y. K. Agarwal, "An algorithm for designing survivable networks," *AT&T Technical Journal*, vol. 68, no. 3, pp. 64–76, 1989.

[63] M. Leitner and G. R. Raidl, "Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks," in *International Workshop on Hybrid Metaheuristics*. Springer, 2008, pp. 158–174.

[64] S. Borne, E. Gourdin, B. Liau, and A. R. Mahjoub, "Design of survivable ip-over-optical networks," *Annals of Operations Research*, vol. 146, no. 1, pp. 41–73, 2006.

[65] G. Dahl and L. Gouveia, "On the directed hop-constrained shortest path problem," *Operations Research Letters*, vol. 32, no. 1, pp. 15–22, 2004.

[66] H. Pirkul and S. Soni, "New formulations and solution procedures for the hop constrained network design problem," *European Journal of Operational Research*, vol. 148, no. 1, pp. 126–140, 2003.

[67] A. R. Mahjoub and S. T. McCormick, "Max flow and min cut with bounded-length paths: complexity, algorithms, and approximation," *Mathematical programming*, vol. 124, no. 1-2, pp. 271–284, 2010.

[68] L. Gouveia, L. Simonetti, and E. Uchoa, "Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs," *Mathematical Programming*, vol. 128, no. 1-2, pp. 123–148, 2011.

[69] L. Gouveia, "Multicommodity flow models for spanning trees with hop constraints," *European Journal of Operational Research*, vol. 95, no. 1, pp. 178–190, 1996.

[70] L. Gouveia and C. Requejo, "A new lagrangean relaxation approach for the hop-constrained minimum spanning tree problem," *European Journal of Operational Research*, vol. 132, no. 3, pp. 539–552, 2001.

[71] G. Dahl, D. Huygens, A. R. Mahjoub, and P. Pesneau, "On the k edge-disjoint 2-hop-constrained paths polytope," *Operations research letters*, vol. 34, no. 5, pp. 577–582, 2006.

[72] L. Gouveia, P. Patrício, and A. de Sousa, "Hop-constrained node survivable network design: An application to mpls over wdm," *Networks and Spatial Economics*, vol. 8, no. 1, pp. 3–21, 2008.

[73] S. Soni, "Hop constrained network design problem with partial survivability," *Annals of Operations Research*, vol. 106, no. 1-4, pp. 181–198, 2001.

[74] F. Bendali, I. Diarrassouba, A. R. Mahjoub, and J. Mailfert, "The k edge-disjoint 3-hop-constrained paths polytope," *Discrete Optimization*, vol. 7, no. 4, pp. 222–233, 2010.

[75] F. Bendali, J. Mailfert, and X. Tang, "Composition of graphs and the hop-constrained path problem," *International Journal of Mathematics in Operational Research*, vol. 4, no. 3, pp. 225–246, 2012.

[76] D. Ronen and Y. Perl, "Heuristics for finding a maximum number of disjoint bounded paths," *Networks*, vol. 14, no. 4, pp. 531–544, 1984.

[77] N. Van der Zijpp and S. F. Catalano, "Path enumeration by finding the constrained k-shortest paths," *Transportation Research Part B: Methodological*, vol. 39, no. 6, pp. 545–563, 2005.