Estudio de algoritmos óptimos y aproximados para el Problema de Localización Dinámica en grafos

Trabajo de Título para optar al título profesional de Ingeniera Civil Matemática

Aline Valentina Zamora Canales

Profesora Guía

Andrea Jiménez
Universidad de Valparaíso

Miembros de la comisión:

Daniel Quiroz
Universidad de Valparaíso
Nicolás Rivera
Universidad de Valparaíso

INSTITUTO DE INGENIERÍA MATEMÁTICA
VALPARAÍSO, 12 DE AGOSTO 2025.

Resumen

En este trabajo se estudia el Problema de Localización Dinámica (PLD) en grafos, formulado por Chung, Graham y Saks [CGS89] en el año 1989. El Problema de Localización Dinámica es un problema de optimización combinatorial enmarcado dentro de las matemáticas discretas, que tiene como sistema un grafo $G = (V, E)$ en donde una fuente ubicada en una cierta posición inicial en algún vértice de G, tiene la habilidad de desplazarse con el objetivo de resolver secuencialmente una lista de requerimientos ubicados en los vértices del grafo minimizando el costo de desplazarse y resolverlos. Estudiamos dos escenarios: el Off-line, en donde la lista de requerimientos por resolver está totalmente disponible desde un inicio, y el On-line, en donde la lista de requerimientos por resolver se va revelando progresivamente y cada desplazamiento de la fuente se debe decidir con información parcial del futuro.

En el contexto Off-line estudiamos propiedades estructurales fundamentales que describen el comportamiento del óptimo para el PLD y diseñamos un algoritmo óptimo, basado en el paradigma de diseño de algoritmos conocido como programación dinámica. Justificamos teóricamente la correctitud de este algoritmo y calculamos la complejidad temporal, resultando en $O(n^2k)$, en donde n es la cantidad de vértices del grafo del sistema, y k es la cantidad de requerimientos por resolver. Implementamos este algoritmo en el lenguaje de programación Phyton para utilizarlo como referencia para comparar el desempeño de algoritmos On-line.

En el caso On-line, siguiendo el trabajo de Chung, Graham y Saks [CGS87, CGS89], introducimos un parámetro en grafos llamado Window Index (Windex), que mide el menor valor de ventana k para el cual existe un algoritmo óptimo para el PLD en G que trabaje tan solo conociendo los siguientes k requerimientos por resolver, es decir, el Windex escencialmente mide qué tanta información necesito conocer del futuro para asegurar la existencia de un algoritmo que encuentre soluciones óptimas. Estudiamos grafos con Windex finito e infinito a partir del cálculo del Windex de clases particulares de grafos. Por otro lado, analizamos la correctitud y complejidad computacional de dos algoritmos:

- Algoritmo Glotón (de ventana 1) que en su diseño se utiliza el paradigma codicioso y garantiza un radio de aproximación 2 para el PLD en cualquier grafo.
- Algoritmo Windex 2 (de ventana 2) que es óptimo para grafos de Windex 2, y sustenta su diseño en la Propiedad de Tripleta de Steiner Única (USTP).

Basándonos en la caracterización de Chung, Graham y Saks [CGS87], probamos que los grafos

con Windex 2 son exactamente los grafos medianos, lo que incluye estructuras como árboles, grillas cuadradas e hipercubos.

Para finalizar, realizamos una parte experimental en donde evaluamos el desempeño del Algoritmo Glotón y del Algoritmo Windex 2 en diferentes grafos, sustentando nuestra elección en el Windex de los grafos seleccionados y variando la densidad de las aristas de los grafos. Las implementaciones de los algoritmos las desarrollamos en el lenguaje de programación Phyton. En el caso del Algoritmo Windex 2, nuestros resultados dan evidencia empírica para la conjetura hecha por Chung, Graham y Saks [CGS89] acerca de la existencia de un algoritmo con ventana 2 y radio de aproximación $3/2$. Por otro lado, a partir de los experimentos realizados, podemos decir que el Algoritmo Windex 2 supera consistentemente al Algoritmo Glotón, especialmente en grafos con Windex 2 o grafos con alta densidad, y que la estructura y densidad del grafo tienen un impacto directo en el desempeño de los algoritmos.

Agradecimientos

En primer lugar, quiero expresar mi gratitud más profunda a mi familia, por su apoyo incondicional, paciencia y confianza a lo largo de toda mi formación universitaria. Su cariño y motivación fueron esenciales para llegar hasta aquí. A mis amistades, gracias por su compañía, por los momentos de alegría y por recordarme que el equilibrio también es parte del éxito académico.

Extiendo mis agradecimientos a todos los profesores y profesoras que me guiaron durante la carrera, por compartir sus conocimientos, motivar la curiosidad y fomentar el pensamiento crítico. A mis compañeros y compañeras de estudio, gracias por las horas compartidas, el intercambio de ideas y el apoyo mutuo en los momentos más exigentes.

De manera especial, agradezco al Proyecto FONDECYT Regular 1220071 (etapas 2024 y 2025) y al Proyecto Núcleo Milenio SODAS ANID-MILENIO-NCN2024-103, cuyo financiamiento hizo posible el desarrollo de esta investigación, brindando los recursos necesarios para llevarla a cabo. Asimismo, agradezco al profesor Lehilton Lelis Chaves Pedrosa de la Universidade Estadual de Campinas por compartir generosamente sus conocimientos sobre el paradigma de programación dinámica, pieza clave para el diseño y análisis presentados en esta tesis.

Asimismo, agradezco profundamente a mi profesora guía, Andrea Jiménez, por su permanente orientación, dedicación y apoyo en cada etapa de este proceso. Su paciencia, rigurosidad académica y compromiso fueron esenciales para dar forma a esta investigación, entregándome siempre observaciones valiosas que enriquecieron tanto el contenido como mi formación profesional y personal.

Finalmente, doy reconocimiento a la Comisión Evaluadora de este trabajo, Andrea Jiménez, Daniel Quiroz y Nicolás Rivera, por su disposición y dedicación para valorar este estudio, así como por el tiempo invertido en su revisión y análisis.

Índice general

Capítulo 1

Problema de Localización Dinámica

Los problemas de localización y búsqueda son desafíos claves en matemáticas discretas dado que logran conectar de manera natural la teoría de grafos, la optimización combinatorial y los algoritmos para dar solución a problemas de la vida cotidiana. Un ejemplo clásico de modelación en grafos es el Problema del Viajante (Travelling Salesman Problem), formalizado de manera sistemática por Lawler et al. [LLRKS85], donde se busca la ruta más corta que recorra un conjunto de ubicaciones exactamente una vez y regrese al punto inicial, combinando optimización combinatorial y diseño de algoritmos.

La optimización secuencial sobre estructuras discretas se ha estudiado en distintos contextos aplicados, como la planificación dinámica de rutas o la asignación de recursos en redes logísticas. Estos problemas requieren de decisiones que deben adaptarse a cambios temporales e información parcial, utilizando técnicas como la descomposición del problema en etapas, la reorganización adaptativa ante secuencias de solicitudes y el uso de grafos para representar relaciones entre elementos [Psa13, OM20, ZPBJ22]. Si bien estas aplicaciones no son el foco del presente estudio, muestran la versatilidad del enfoque secuencial en distintos ámbitos como la informática teórica, la gestión de datos y la ciencia de redes. Por lo tanto, estudiar estrategias para optimizar el acceso en contextos secuenciales no solo tiene aplicaciones concretas, sino que además plantea interrogantes fundamentales sobre la estructura y el comportamiento de sistemas dinámicos.

En este trabajo de tesis se estudia un Problema de Localización Dinámica (PLD) introducido por Chung, Graham y Saks en [CGS89], en el cual se procesan requerimientos que ocurren de manera secuencial en los nodos de un grafo y deben ser atendidas. Este problema surge de la necesidad de tomar desiciones en tiempo real con información parcial (On-Line) buscando soluciones óptimas o cercanas a ella, lo que se conoce como modelación de instancias secuenciales en contextos dinámicos. La formulación del problema describe el concepto de Windex –un parámetro que evalúa propiedades estructurales de los grafos– y detalla métodos para determinar si su valor es finito, infinito o un entero específico. Este análisis es clave para comprender las relaciones dinámicas en grafos y su impacto en el PLD.

El PLD se relaciona con problemas clásicos de ubicación de recursos como el k-median problem y el uncapacitated/capacitated resource/facility location problem. Estos problemas están

orientados a determinar ubicaciones óptimas para una o más fuentes en una red, minimizando el costo asociado a proveer servicios a un conjunto de vértices designados como clientes utilizando la fuente más cercana. La investigación en esta área se ha centrado en diseñar algoritmos factibles que determinen la solución óptima y la efectividad de estrategias para encontrar soluciones aproximadas [Wil10].

Por otro lado, lo que hace interesante al PLD es su interpretación en un contexto On-Line, en donde la información completa de la instancia no está disponible desde el inicio sino que se revela progresivamente con el tiempo, planteando interrogantes sobre la existencia de estrategias óptimas que puedan operar bajo un conocimiento limitado del futuro. Se asume entonces que se conoce el grafo, la posición inicial de la fuente y un conjunto limitado de requerimientos que llegan secuencialmente a medida que se resuelven. A continuación, se procederá a formalizar estos conceptos.

1.1. Definición del Problema

Para comprender adecuadamente el Problema de Localización Dinámica (PLD), introducido por Chung, Graham y Saks [CGS89], es necesario establecer primero ciertos conceptos fundamentales. En particular, definimos lo que se entiende por una instancia del Problema de Localización Dinámica y por algoritmos que permiten abordar el problema desde un enfoque computacional. En general, una instancia de un problema computacional representa la información de entrada que contiene los elementos necesarios para resolver el problema bajo las condiciones y restricciones impuestas. A partir de esta entrada, un algoritmo actúa como un procedimiento finito y bien definido que transforma dicha información en una solución, mediante una secuencia ordenada de pasos computacionales.

Dentro del estudio de algoritmos, se pueden diferenciar dos categorías según el momento en que la información de entrada, es decir la instancia, está disponible: los algoritmos Off-Line y los algoritmos On-Line. Un algoritmo Off-Line opera con total conocimiento de la información desde el inicio, es decir, con la instancia completa. En cambio, un algoritmo On-Line toma decisiones en función de la información disponible en el instante actual, sin acceso completo al futuro del problema, es decir con conocimiento parcial de la instancia. Esta distinción es particularmente relevante en problemas como el PLD, donde el comportamiento del sistema puede depender críticamente de la disponibilidad temporal de los datos.

El Problema de Localización Dinámica es un problema de optimización combinatorial que recibe como entrada un grafo $G = (V, E)$ en donde el conjunto de vértices V representa ubicaciones y las aristas representan conexiones entre las ubicaciones, una fuente que comienza en una posición inicial $s_0 \in V(G)$ y una lista de requerimientos $r_1 r_2 ... r_n \in V(G)$ ubicados en los vértices del grafo y que deben resolverse sequencialmente. La fuente del sistema tiene la habilidad de moverse de un vértice u a otro v con costo $d_G(u, v)$, y la resolución de cada requerimiento r tiene costo $d_G(r, s)$, en donde s es el vértice en donde se encuentra la fuente justo antes de resolver el

requerimiento r, y para $x, y \in V(G)$, la notación $d_G(x, y)$ denota la distancia entre x e y en el grafo G, es decir la cantidad de aristas de un camino más corto que conecta x e y en G; cuando el grafo es claro dentro del contexto, usamos $d(x, y)$ en vez de $d_G(x, y)$. El objetivo de este problema es encontrar una secuencia de ubicaciones para la fuente de manera que el costo de resolver todos los requerimientos secuencialmente sea mínimo. En lo que sigue, definimos formalmente el PLD.

Definición 1 (Problema de Localización Dinámica (PLD))**.** *Dado un grafo G, una posición inicial de la fuente $s_0 \in V(G)$, y una lista de vértices $r_1 r_2 ... r_n \in V(G)$ llamados* requerimientos*, el problema de localización dinámica consiste en determinar alguna secuencia de posiciones de la fuente $s_1 ... s_n \in V(G)$ que minimice la siguiente función:*

$$C(s_1 ... s_n; s_0 r_1 ... r_n) = \sum_{i=1}^{n} d(s_{i-1}, s_i) + d(s_i, r_i) \tag{1.1}$$

La función C es llamada función objetivo *del PLD y se dirá que una secuencia $s'_1 ... s'_n$ es óptima cuando la secuencia minimiza la función objetivo, es decir:*

$$OPT(s_0 r_1 ... r_n) = C(s'_1 ... s'_n; s_0 r_1 ... r_n) = \min_{s_1 ... s_n} C(s_1 ... s_n; s_0 r_1 ... r_n) \tag{1.2}$$

Diremos que un algoritmo $\mathcal{A}$ resuelve el PLD en el grafo G si dada cualquier instancia $s_0 r_1 \dots r_n$, el algoritmo $\mathcal{A}$ determina alguna secuencia de vértices $s_1 s_2 \dots s_n$ que minimiza la función objetivo definida en la ecuación (1.1).

Como se ha mencionado anteriormente, en este trabajo se estudian dos tipos de algoritmos para el PLD: algoritmos Off-Line y algoritmos On-Line. La diferencia entre ambos radica en la cantidad de información con que cuenta el algoritmo para producir una respuesta. En este sentido, los algoritmos Off-Line son aquellos que reciben la instancia del problema completa al inicio de la ejecución del algoritmo, y que pueden procesar toda la información para obtener una salida. Por otro lado, se tienen los algoritmos On-Line, siendo aquellos que reciben solo una parte de la instancia del problema, debiendo procesar información incompleta para producir soluciones parciales, y en donde la instancia completa se va revelando progresivamente en el tiempo, a medida que el algoritmo produce sus soluciones parciales.

Un ejemplo de algoritmo Off-Line que resuelve el PLD en un grafo G sería la siguiente estrategia exhaustiva: dada la instancia $s_0 r_1 \dots r_n$, el algoritmo calcula el costo C de cada las posibles posiciones de la fuente $s_1 \dots s_n$, que son $|G|^n$ distintas, y tomar una de aquellas que minimiza la función objetivo. Es claro que este algoritmo efectivamente resuelve el PLD en todo grafo, sin embargo, un algortimo de tal tipo resulta inviable en la práctica. Luego, surge la interrogante de la existencia de algoritmos eficientes que resuelvan el problema. Es así como nos encontramos frente al primer objetivo de este trabajo de tesis, el desarrollo de algoritmos Off-Line eficientes que resuelvan el PLD, lo cual estudiamos en el Capítulo 2.

Por otro lado, para el PLD en el escenario On-Line, tenemos un grafo G, la posición inicial de la fuente y un conjunto limitado de requerimientos que llegan secuencialmente a medida que se resuelven. La primera interrogante es la existencia de algoritmos óptimos que puedan operar bajo

este conocimiento limitado del futuro. Con el objetivo de poder describir el contexo On-Line para el PLD vamos a definir el concepto de Windex, introducido por Chung, Graham y Saks [CGS89].

Definición 2. *Sea $\mathcal{A}$ un algoritmo que resuelve el PLD en el grafo G. Para $k \in \mathbb{N}, k > 1$, decimos que $\mathcal{A}$* trabaja dentro de una ventana *k si la elección de s_i (posición i-ésima de la fuente) sólo depende de s_{i-1} y los requrimientos j-ésimos r_j, con $j < i + k$.*

Definición 3 (Window Index / Windex)**.** *El* Window Index *(también llamado* Windex*) de un grafo G se denota por $WX(G)$ y es el menor $k \in \mathbb{N}$ tal que existe un algoritmo $\mathcal{A}$ que resuelve el PLD en G que trabaja dentro de una ventana k. Si tal algoritmo $\mathcal{A}$ no existe, entonces $WX(G) = \infty$.*

Como fue mencionado anteriormente, el concepto de Windex fue introducido por primera vez por Chung, Graham y Saks en [CGS89], en donde además de introducir el concepto de Windex, investigan la existencia de algoritmos óptimos con Windex finito para distintas clases de grafos. En particular, dan ejemplos de grafos con Windex infinito –grafos para los cuales no existe ningún algoritmo On-Line que resuelva el PLD para un valor arbitrario de ventana–, estudian el Windex para clases particulares de grafos, el comportamiento del Windex para el producto de grafos y caracterizan los grafos con Windex 2. En el Capítulo 3 de este trabajo de tesis, hacemos una introducción al parámetro Windex basado en el trabajo de Chung, Graham y Saks en [CGS87]. Luego, en el Capítulo 4 hacemos un estudio completo de los grafos con Windex 2.

En [CGS89], Chung, Graham y Saks logran caracterizar la familia de grafos con Windex finito. Con esto, muestran que la mayoría de los grafos tienen Windex infinito, y es entonces interesante, enfocar los esfuerzos en el diseño de algoritmos que trabajen dentro de una ventana k, y que entreguen soluciones aproximadas para el PLD.

Definición 4. *Sea G un grafo, y $\rho \geq 1$. Decimos que un algoritmo $\mathcal{A}$ aproxima el PLD en el grafo G con un radio ρ, si para cualquier instancia $s_0 r_1 \dots r_n$, el algoritmo $\mathcal{A}$ determina alguna secuencia de vértices $s_1 s_2 \dots s_n$ que satisface*

$$C_{\mathcal{A}}(s_0 r_1 \dots r_n) = C(s_1 ... s_n; s_0 r_1 ... r_n) \leq \rho \cdot \mathrm{OPT}(s_0 r_1 \dots r_n).$$

Así como en el caso de algoritmos que resuelven el PLD en G, podemos definir algoritmos aproximados que trabajan dentro de un cierto parámetro de ventana.

Definición 5. *Sea $\mathcal{A}$ un algoritmo que aproxima el PLD en el grafo G con un radio ρ. Para $k \in \mathbb{N}$ $k \geq 1$, decimos que $\mathcal{A}$* trabaja dentro de una ventana *k si la elección de s_i (posición i-ésima de la fuente) sólo depende de s_{i-1} y los requrimientos j-ésimos r_j, con $j < i + k$.*

En [CGS89], los autores conjeturan que existen algoritmos que trabajan dentro de una ventana k y que aproximan el PLD con un radio de aproximación que decrece en función de k.

Conjetura 1. *Sea G un grafo y k un entero $k \geq 1$. Entonces, existe un algoritmo $\mathcal{A}$ que aproxima el PLD en G con un radio*

$$\rho = \left(1 + \frac{1}{k}\right)$$

y que trabaja dentro de una ventana k.

En el Capítulo 3 de este trabajo, mostramos un algoritmo que trabaja dentro de una ventana 1 y que aproxima el PLD con radio 2. En el Capítulo 5, damos evidencia empírica para la Conjetura 1 en el caso $k = 2$ utilizando los algoritmos desarrollados en el Capítulo 2 y el Capítulo 4. Además, con sustento en los experimentos realizados, proponemos mejores radios de aproximación, tanto para $k = 1$ como para $k = 2$ en algunas clases de grafos.

1.2. Ejemplos

Una vez definidos los conceptos fundamentales del Problema de Localización Dinámica, es útil ilustrar cómo distintas estrategias pueden conducir a soluciones con rendimientos muy diferentes. Para ello, en esta sección se presentan ejemplos sobre un mismo grafo e instancia, aplicando tres enfoques algorítmicos distintos. Esta comparación permite evidenciar de forma concreta la motivación por diseñar algoritmos óptimos para el PLD y justifica el estudio de estrategias como el uso del parámetro Window Index.

Para las tres estrategias, se entrega un mismo grafo, nodo fuente s_0 y lista de requerimientos a realizar, es decir, la instancia. Lo que se busca, es minimizar el costo de resolver todos los requerimientos $r_1r_2r_3r_4$. En donde, recordemos, el costo se define por la ecuación (1.1). Para esto, se deben seguir las siguientes reglas:

1. Los requerimientos se deben resolver secuencialmente.
2. En cada paso se tiene la habilidad de mover la fuente de s_{i-1} a s_i con un costo asociado $d(s_{i-1}, s_i)$.
3. El costo de resolver el requerimiento r_i es $d(s_i, r_i)$

Se intentará obtener una secuencia $s_1s_2s_3s_4$ tal que la siguiente expresión sea mínima:

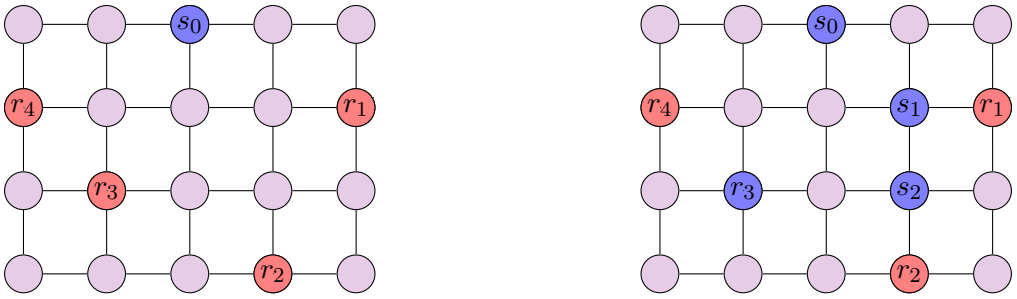
$$\sum_{i=1}^{4} d(s_{i-1}, s_i) + d(s_i, r_i).$$

Se presenta a continuación distintas perspectivas (óptimas y no óptimas) para estudiar posibles resultados del siguiente grafo.

Para el primer ejemplo se observa la Figura 1.1(a) en donde se tiene el caso en el cual la fuente permanece en la misma posición inicial $s_0 = s_1 = s_2 = s_3 = s_4$, entonces el costo de resolver los requerimientos es el siguiente:

$$\begin{aligned}\sum_{i=1}^{4} (d(s_{i-1}, s_i) + d(s_i, r_i)) &= d(s_0, r_1) + d(s_0, r_2) \\ &\quad + d(s_0, r_3) + d(s_0, r_4) \\ &= 13\end{aligned}$$

Para el segundo ejemplo se mantiene la Figura 1.1(a) pero ahora se intenta mejorar la estrategia de la secuencia de posiciones de la fuente, siguiendo una estrategia “glotona” que consiste



(a) (b)

Figura 1.1: Grafos representados para cada ejemplo con nodos azules para las posiciones de la fuente y nodos rojos para los requerimientos.

en mover la fuente a la misma posición del requerimiento i-ésimo de manera que $s_1 = r_1$, $s_2 = r_2$, $s_3 = r_3$ y $s_4 = r_4$. Cabe destacar que esta estrategia hace referencia a un algoritmo que trabaja dentro de una ventana 1 dado que utiliza solamente el requerimiento r_i para determinar la posición s_i de la fuente. Se tiene entonces que:

$$\begin{aligned}\sum_{i=1}^{4}(d(s_{i-1},s_i)+d(s_i,r_i)) &= d(s_0,r_1)+d(r_1,r_2)+d(r_2,r_3)\\ &\quad + d(r_3,r_4)\\ &= 11\end{aligned}$$

Para el último ejemplo se utiliza una estrategia que entrega el óptimo, que trabaja dentro de una ventana 2, y que será descrita en el Capítulo 4). Para la Figura 1.1(b), se observa que la posición de la fuente s_3 y s_4 están en la misma posición que r_3. Con esta estrategia, tenemos que:

$$\begin{aligned}\sum_{i=1}^{4}(d(s_{i-1},s_i)+d(s_i,r_i)) &= d(s_0,s_1)+d(s_1,r_1)+d(s_1,s_2)+d(s_2,r_2)\\ &\quad + d(s_3,r_3)+d(s_4,r_4)\\ &= 9\end{aligned}$$

Los ejemplos analizados revelan cómo distintas estrategias en la elección de posiciones para las fuentes generan variaciones en el costo total de resolver los requerimientos secuencialmente. Estas diferencias sustanciales en el desempeño de cada metodología evidencian la importancia de estudiar sistemáticamente el Problema de Localización Dinámica, analizando comparativamente las diversas aproximaciones y sus criterios de optimalidad.

1.3. Organización y contribuciones

El objetivo general del proyecto es estudiar el PLD tanto en su versión Off-Line (información completa) y On-Line (información incompleta). Para ello, se diseñarán y analizarán algoritmos

eficaces para ambos contextos, se explorarán propiedades estructurales (como el Window index) y se caracterizará la complejidad computacional del problema. Estos resultados no solo avanzarán la teoría subyacente, sino que también ofrecerán herramientas para aplicaciones prácticas en optimización dinámica. La metodología propuesta incluye una amplia revisión de la literatura, el diseño y la implementación de algoritmos, y la evaluación de su desempeño en diferentes escenarios.

En el Capítulo 2 se estudia el PLD en su versión Off-Line. Se presentan propiedades fundamentales de las secuencias óptimas, formulando el problema mediante una estructura recursiva que permite el uso del paradigma de Programación Dinámica. Se propone el Algoritmo PLD Off-Line, se demuestra su correctitud y se analiza su complejidad temporal, obteniendo un método eficaz para el cálculo exacto del óptimo.

En el Capítulo 3 se aborda el PLD en su versión On-Line, introduciendo formalmente el concepto de Windex y revisando resultados conocidos sobre su valor en distintas familias de grafos. Se presentan casos con Windex finito e infinito, y se diseña un algoritmo glotón que trabaja con ventana de tamaño uno, probando que aproxima el PLD con un radio de aproximación igual a 2.

El Capítulo 4 se dedica al estudio de grafos con Windex dos. Se demuestra que los grafos medianos son exactamente aquellos con $WX(G) = 2$, describiendo la estrategia USTP como algoritmo óptimo en este contexto. Se presentan resultados estructurales (producto cartesiano, caracterización) y se implementa el Algoritmo Windex 2, analizando su correctitud y complejidad.

En el Capítulo 5 se presentan las implementaciones y experimentos. Se realizan dos estudios: influencia del Windex en el desempeño de los algoritmos, evaluando árboles, grafos completos y grafos con Windex infinito; y el impacto de la densidad de aristas en grafos aleatorios. En ambos casos se compara el costo y el radio de aproximación de los algoritmos PLD Off-Line, Glotón y Windex 2. Los resultados muestran que Windex 2 supera consistentemente al Glotón, especialmente en grafos con $WX(G) = 2$ o alta densidad, y que la estructura del grafo influye notablemente en el rendimiento de los algoritmos On-Line.

Entre las principales contribuciones de este trabajo se destacan:

- La formulación detallada de propiedades estructurales del PLD que sustentan el diseño de algoritmos eficientes.
- El desarrollo y análisis de un algoritmo exacto Off-Line basado en Programación Dinámica.
- El diseño de algoritmos aproximados para ventanas $k = 1$ y $k = 2$, junto con la demostración de sus radios de aproximación teóricos.
- La implementación de un plan experimental que permite contrastar el rendimiento teórico y empírico de los algoritmos, identificando patrones de comportamiento según la estructura del grafo.
- Evidencia empírica a favor de la conjetura planteada por Chung, Graham y Saks [CGS89] respecto a la existencia de algoritmos On-Line con radio de aproximación $(1 + 1/k)$, en particular para $k = 2$.

Capítulo 2

Problema de Localización Dinámica Off-line

Este capítulo está dedicado al estudio del PLD en su versión Off-line. Para comenzar, en la Sección 2.1 enunciamos y demostramos algunas propiedades fundamentales de las secuencias óptimas en el PLD, concluyendo con el Teorema 4, que sienta las bases para el diseño de un algoritmo eficiente que resuelve el PLD Off-line en cualquier grafo. En la Sección 2.2 hacemos una discusión acerca del paradigma de diseño de algoritmos conocido como programación dinámica, para luego presentar el pseudocódigo de un algoritmo Off-line que resuelve el PLD en grafos, diseñado en base al paradigma de programación dinámica. Además del pseudocódigo (Algoritmo 1) ilustramos el funcionamiento del algoritmo con un ejemplo de ejecución para una entrada particular. Para finalizar este capítulo, en Sección 2.3 se discute el análisis de la correctitud y el tiempo de ejecución del Algoritmo PLD Off-line (Algoritmo 1). Más tarde, en el Capítulo 5, se utilizará el Algoritmo PLD Off-line desarrollado en este capítulo para realizar un estudio empírico de la Conjectura 1 en el caso $k = 2$, y estudiar el desempeño en la práctica para ciertas clases de grafos, de dos algoritmos aproximados que se presentarán en Capítulos 3 and 4.

2.1. Propiedades fundamentales

Antes de abordar el diseño y el análisis de algoritmos para el PLD Off-line, es crucial establecer ciertos resultados teóricos que sirven de base para su estudio. En esta sección presentamos varios resultados enunciados en [CGS89], cuyas formulaciones ofrecen propiedades claves sobre el comportamiento de las secuencias óptimas del PLD para el diseño eficaz de un algoritmo Off-line.

Si bien, como mencionamos, los enunciados de estos resultados ya han sido planteados en la literatura, no se dispone de demostraciones explícitas en las fuentes consultadas, por lo que en este trabajo se desarrollan pruebas propias que sustentan tales afirmaciones. Además, debido a la relevancia estructural del segundo resultado (Teorema 4) originalmente propuesto como

una proposición en [CGS89], se eleva su categoría a teorema para enfatizar su importancia en el análisis posterior de la correctitud del algoritmo principal, pues da muestra de la naturaleza recursiva de la función objetivo.

Primero que todo, es importante recordar una propiedad estructural básica pero muy útil de los grafos, conocida como desigualdad triangular. El conjunto de los vértices de un grafo forma un espacio métrico, considerando como distancia entre dos vértices la cantidad de aristas en un camino más corto. En este contexto, usaremos la siguiente proposición como una herramienta clave para las demostraciones a lo largo de este trabajo de tesis.

Proposición 2. *Sea G un grafo, y sean $x, y, v \in V(G)$. Entonces*

$$d(x, v) + d(v, y) \geq d(x, y).$$

Además, la igualdad se alcanza si y solo si v pertenece a un camino más corto entre x y y.

La proposición anterior será utilizada de manera reiterada en las pruebas siguientes, ya que permite descomponer caminos y establecer cotas inferiores sobre combinaciones de distancias. A continuación, se enuncian algunas propiedades básicas que cumple la función objetivo OPT del PLD, las cuales serán clave para construir y justificar algoritmos en los capítulos siguientes.

Proposición 3.

I) *OPT*$(s_0) = 0$

II) *OPT*$(s_0 r_1) = d(s_0, r_1)$

III) *OPT*$(s_0 r_1 r_2) = \min_{v \in V}\{d(v, s_0) + d(v, r_1) + d(v, r_2)\}$

IV) *Sea α_i una secuencia de requerimientos, con $i \in \{1, 2, ..., k\}$. Para cualquier secuencia $\alpha_1, \alpha_2, \ldots, \alpha_k$, se cumple que*

$$\textit{OPT}(\alpha_1 \alpha_2 \ldots \alpha_k) \geq \textit{OPT}(\alpha_1) + \textit{OPT}(\alpha_2) + \cdots + \textit{OPT}(\alpha_k)$$

Demostración.

I) OPT$(s_0) = 0$

Trivial para $k = 0$ puesto que no hay requerimientos que cumplir. La posición de la fuente inicial s_0 no necesita desplazarse por lo que no hay función objetivo a minimizar.

II) OPT$(s_0 r_1) = d(s_0, r_1)$

Consideramos un requerimiento a cumplir, por lo que existe s_1 que resuelve r_1, entonces

$$C(s_1; s_0 r_1) = d(s_0, s_1) + d(s_1, r_1).$$

Tenemos por desigualdad triangular que $d(s_0, r_1) \leq d(s_0, s_1) + d(s_1, r_1)$. Luego:

$$\text{OPT}(s_0 r_1) = \min_{s_1} C(s_1; s_0 r_1) = d(s_0, r_1).$$

III) $\text{OPT}(s_0r_1r_2) = min_{v\in V}\{d(v,s_0)+d(v,r_1)+d(v,r_2)\}$

Consideramos 2 requerimientos a cumplir, por lo que existe s_1 que resuelve r_1 y s_2 que resuelve r_2. Sean s_0, s_1, s_2, r_1, r_2 vértices del grafo.
Definimos el costo total de resolver los requerimientos con las posiciones de la fuente $x, y \in V(G)$ como:

$$C(xy; s_0r_1r_2) = d(s_0,x) + d(x,r_1) + d(x,y) + d(y,r_2) \quad \forall x, y \in V(G)$$

Definimos la función óptima como el mínimo costo total de la forma:

$$\text{OPT}(s_0r_1r_2) = \min_{x,y\in V(G)}\{d(s_0,x) + d(x,r_1) + d(x,y) + d(y,r_2)\} = \min_{x,y\in V(G)} C(xy; s_0r_1r_2)$$

Sea $s_0s_1s_2$ secuencia óptima que minimiza el costo total. Es decir:

$$\text{OPT}(s_0r_1r_2) = C(s_1s_2; s_0r_1r_2)$$

Notamos que $d(s_2,s_0) \leq d(s_0,s_1) + d(s_1,s_2)$, por lo tanto

$$d(s_2,s_0) + d(s_1,r_1) + d(s_2,r_2) \leq d(s_0,s_1) + d(s_1,r_1) + d(s_1,s_2) + d(s_2,r_2), \quad \forall s_1, s_2 \in V(G)$$

En particular,

$$C(s_1s_2; s_0r_1r_2) \leq \min_{v\in V(G)}\{d(v,s_0)+d(v,r_1)+d(v,r_2)\} \leq d(s_0,s_1)+d(s_1,r_1)+d(s_1,s_2)+d(s_2,r_2)$$

Por desigualdad triangular:

$$d(s_2,s_0) + d(s_1,r_1) + d(s_1,r_2) \leq d(s_0,s_1) + d(s_1,r_1) + d(s_1,s_2) + d(s_2,r_2)$$

Se concluye que $s_1 = s_2$ y $C(s_1s_2; s_0r_1r_2) = \min_{v\in V(G)}\{d(v,s_0) + d(v,r_1) + d(v,r_2)\}$

IV) Para cualquier secuencia de secuencias $\alpha_1, \alpha_2, \ldots, \alpha_k$, queremos probar que

$$\text{OPT}(\alpha_1\alpha_2\ldots\alpha_k) \geq \text{OPT}(\alpha_1) + \text{OPT}(\alpha_2) + \cdots + \text{OPT}(\alpha_k)$$

Sea s_0 fuente inicial y $\Theta = \{\alpha_1 \cup \alpha_2 \cup \cdots \cup \alpha_k\}$ la secuencia total de requerimientos. Para cada $i \in \{1,\ldots,k\}$, el $\text{OPT}(\alpha_i)$ minimiza el costo de α_i como problema independiente.

Sea Θ^* solución de $\text{OPT}(\Theta)$, es decir:

$$\text{OPT}(\Theta) = C(\Theta^*)$$

Consideramos que se asocia la secuencia de fuentes de Θ^* a los requerimientos de α_i, para cada $i \in \{1,\ldots,k\}$. Esto implica que los subrutas de Θ^* no son necesariamente óptimas para α_i, pero sí factible con costo C_i. Entonces:

$$\text{OPT}(\alpha_i) \leq C_i.$$

Luego:

$$\sum_{i=1}^{k} \text{OPT}(\alpha_i) \leq \sum_{i=1}^{k} C_i = C(\Theta^*) = \text{OPT}(\Theta) \Rightarrow \text{OPT}(\Theta) \geq \text{OPT}(\alpha_1)+\text{OPT}(\alpha_2)+\cdots+\text{OPT}(\alpha_k)$$

□

La proposición anterior permite establecer algunos casos base y propiedades estructurales del valor óptimo del PLD en instancias pequeñas o particionadas. A partir de estas ideas, se puede formular un resultado más general que caracteriza de manera recursiva el comportamiento óptimo del problema.

El siguiente teorema, también planteado en [CGS89], será central en el desarrollo de un algoritmo óptimo y eficiente, ya que permite descomponer instancias del PLD en subproblemas más pequeños, facilitando el uso del paradigma de diseño de algoritmos llamado programación dinámica.

Teorema 4. *Si* $s_0, r_1, ..., r_k \in V(G)$ *y* $1 \leq j \leq k$*, entonces*

$$\mathit{OPT}(s_0 r_1 ... r_k) = \min_{v \in V} \{d(v, r_j) + \mathit{OPT}(s_0 r_1 ... r_{j-1} v) + \mathit{OPT}(v r_{j+1} ... r_k)\} \tag{2.1}$$

y en particular

$$\mathit{OPT}(s_0 r_1 ... r_k) = \min_{v \in V} \{d(s_0, v) + d(s_1, v) + \mathit{OPT}(v r_2 ... r_k)\}$$

Demostración. Primero se demuestra la desigualdad $\geq$. Supongamos $s_1, ..., s_k$ una secuencia óptima de fuentes. Por desigualdad triangular, se cumple $d(r_j, s_j) \leq d(v, s_j) + d(r_j, v)$ para cualquier $v \in V$. Entonces,

$$\begin{aligned}
\mathrm{OPT}(s_0 r_1 ... r_k) &= \sum_{i=1}^{k} d(s_{i-1}, s_i) + d(r_i, s_i) \\
&= \sum_{i=1}^{j-1} d(s_{i-1}, s_i) + d(r_i, s_i) + d(s_{j-1}, s_j) + d(r_j, s_j) + d(s_j, s_{j+1}) + d(r_{j+1}, s_{j+1}) \\
&+ \sum_{i=j+2}^{k} d(s_{i-1}, s_i) + d(r_i, s_i)
\end{aligned}$$

Sea $z = s_j$, sabemos que $d(z, s_j) = 0$, entonces

$$\begin{aligned}
\mathrm{OPT}(s_0 r_1 ... r_k) &= \sum_{i=1}^{j-1} d(s_{i-1}, s_i) + d(r_i, s_i) + d(s_{j-1}, z) + d(r_j, z) + d(z, s_{j+1}) + d(r_{j+1}, s_{j+1}) \\
&+ \sum_{i=j+2}^{k} d(s_{i-1}, s_i) + d(r_i, s_i) \\
&\geq d(z, r_j) + \mathrm{OPT}(s_0 r_1 ... r_{j-1} z) + \mathrm{OPT}(z r_{j+1} ... r_k) \\
&\geq \min_{v} \{d(v, r_j) + \mathrm{OPT}(s_0 r_1 ... r_{j-1} v) + \mathrm{OPT}(v r_{j+1} ... r_k)\}
\end{aligned}$$

Ahora se demuestra la desigualdad $\leq$. Sea $x \in V$ que minimiza

$$d(x, r_j) + \mathrm{OPT}(s_0, r_1, \ldots, r_{j-1}, x) + \mathrm{OPT}(x, r_{j+1}, \ldots, r_k)$$

Sea $y_1 y_2 \dots y_j$ una secuencia de fuentes óptima que minimiza OPT$(s_0 r_1 \dots r_{j-1} v)$ y sea $y_{j+1} \dots y_k$ una secuencia de fuentes óptima que minimiza OPT$(v r_{j+1} \dots r_k)$. Tenemos que:

$$
\begin{aligned}
& d(x, r_j) + \text{OPT}(s_0 r_1 \dots r_{j-1} x) + \text{OPT}(x r_{j+1} \dots r_k) \\
& = d(x, r_j) + d(s_0, y_1) + d(y_1, r_1) + \sum_{i=2}^{j-1} (d(y_{i-1}, y_i) + d(y_i, r_i)) \\
& + \underbrace{d(y_{j-1}, y_j) + d(y_j, x)}_{\text{des. triangular}} + d(x, y_{j+1}) + d(y_{j+1}, r_{j+1}) + \sum_{i=j+2}^{k} (d(y_{i-1}, y_i) + d(y_i, r_i))
\end{aligned}
$$

$$
\begin{aligned}
& \geq d(s_0, y_1) + d(y_1, r_1) + \sum_{i=2}^{j-1} (d(y_{i-1}, y_i) + d(y_i, r_i)) + d(y_{j-1}, x) + d(x, r_j) \\
& + d(x, y_{j+1}) + d(y_{j+1}, r_{j+1}) + \sum_{i=j+2}^{k} (d(y_{i-1}, y_i) + d(y_i, r_i)) \\
& \geq \min_{s_1 \dots s_k} \left(\sum_{i=1}^{k} d(s_{i-1}, s_i) + d(s_i, r_i) \right) = \text{OPT}(s_0 r_1 \dots r_k)
\end{aligned}
$$

□

2.2. Programación Dinámica y un Algoritmo Off-line

La Programación Dinámica es un paradigma de diseño de algoritmos que permite resolver problemas complejos mediante su descomposición en problemas más pequeños, almacenando sus resultados para evitar cálculos redundantes. Este paradigma se basa en etapas estructuradas: primero se formula el problema identificando decisiones relevantes y variables de estado, luego se define la relación de recurrencia para expresar la solución a partir de soluciones previas, se identifican los casos base (instancias que no requieren decisiones anteriores), y finalmente se elige una estrategia de resolución (top-down o bottom-up). Se aplica en contextos donde la solución óptima global se construye a partir de soluciones óptimas de sus subproblemas evaluadas múltiples veces.

El diseño de un algoritmo basado en programación dinámica sigue estas etapas estructuradas que permiten descomponer el problema, identificar sus componentes esenciales y construir una solución eficiente. Estas etapas son fundamentales para resolver problemas que cumplan con la estructura mencionada. En el PLD Off-line, cada subproblema puede definirse como el costo mínimo de resolver los primeros n requerimientos, incluyendo costos acumulados hasta el tiempo anterior, costo de movimiento entre posiciones y costo de resolver el requerimiento actual, con el caso base siendo la posición inicial de la fuente y el primer requerimiento. Lo anterior es sustentado por el Teorema 4.

En este trabajo de tesis, para abordar el problema de resolver el PLD Off-line hemos utilizado programación dinámica, dada la estructura secuencial y multietapa, donde el resultado óptimo considera consecuencias futuras al depender directamente de decisiones tomadas en tiempos anteriores, reflejando la subestructura óptima. La superposición de subproblemas se evidencia en la repetición de búsqueda de costo mínimo para tiempos específicos, permitiendo evaluar problemas con horizonte temporal extenso más eficientemente que técnicas exhaustivas. Dada la recursividad construida con el Teorema 4 mencionado en la sección anterior, es apropiado adoptar una estrategia Bottom-Up de atrás hacia adelante, que calcula el valor óptimo global mediante minimizaciones entre distancias sin redundancias gracias a la memoria del enfoque. La estrategia backward optimiza el cálculo y refleja la dependencia temporal inherente al PLD Off-line, justificando su elección como base del algoritmo propuesto.

Las ventajas de este enfoque son notables, pero presenta limitaciones como alto consumo de memoria cuando el número de estados crece exponencialmente, o la complejidad de formular subproblemas y relaciones de recurrencia. Como veremos, en el PLD Off-line esto se refleja en matrices de costos que crecen con tiempos y posiciones posibles, afectando potencialmente la escalabilidad. Sin embargo, pese a estas limitaciones, la programación dinámica sigue siendo una elección sólida para el PLD Off-line por su equilibrio entre exactitud, eficiencia y claridad algorítmica.

En resumen, su uso en el PLD Off-line responde tanto a necesidades prácticas de eficiencia computacional como a una correspondencia estructural profunda entre paradigma y problema. Esta relación justifica el enfoque adoptado y constituye la base del algoritmo presentado, que será formalizado mediante un modelo matemático detallado y pseudocódigo estructurado para reflejar tanto la teoría general de Programación Dinámica como las particularidades del PLD.

Algoritmo PLD Off-line

Una vez establecido el comportamiento de las secuencias óptimas del PLD mediante resultados teóricos (Teorema 4), en esta sección se presenta un algoritmo que permite resolver el PLD en su versión Off-line. Como dijimos anteriormente, este algoritmo se basa en una estrategia de programación dinámica, aprovechando la caracterización recursiva de la función objetivo presentada anteriormente.

Se incluye a continuación el pseudocódigo que describe el procedimiento computacional para hallar una secuencia óptima de posiciones de la fuente, junto con el valor mínimo asociado al costo total. Posteriormente, se presenta un ejemplo ilustrativo sobre un grafo simple, el cual permite visualizar paso a paso la obtención del valor óptimo y la construcción de la solución óptima. Este ejemplo facilitará la comprensión de la ejecución del algoritmo y evidencia su efectividad para resolver instancias concretas del PLD. Observamos que en la entrada del Algoritmo 1 se considera la matriz A de las distancias del grafo de entrada G, en donde para $V(G) = \{v_1, v_2, \ldots, v_n\}$, la entrada i, j de la matriz A corresponde a la distancia en G entre v_i y v_j, es decir $d_G(v_i, v_j)$.

Algoritmo 1 Algoritmo PLD Off-line

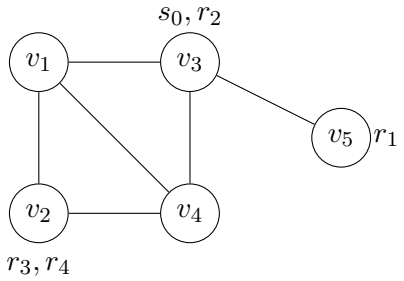
Entrada: $G = (V, E)$; matriz de distancias $A_{n\times n}$; requerimientos $s_0 r_1 \dots r_k$
Salida: Secuencia óptima $s_1 s_2 \dots s_k$ que minimiza $\sum_{i=1}^{k}[d(s_{i-1}, s_i) + d(r_i, s_i)]$

Memoria $\leftarrow M_{n\times k\times 2}(0)$
$OPT(vr_k) \leftarrow d(v, r_k)$
for $i \leftarrow 1$ **to** n **do**
    Memoria$[i][k] \leftarrow [OPT(v_i r_k), v_i]$
end for
for $j \leftarrow k-1$ **downto** 2 **do**
    for $i \leftarrow 1$ **to** n **do**
        Memoria$[i][j][1] \leftarrow \min_{\ell\in\{1,\dots,n\}}\{d(v_i, v_\ell) + d(r_j, v_\ell) +$ Memoria$[\ell][j+1][1]\}$
        Memoria$[i][j][2] \leftarrow v_\ell$ tal que Memoria$[i][j][1]$ es mínima
    end for
end for
$r_1 \leftarrow$ primer requerimiento
for $i \leftarrow 1$ **to** n **do**
    Memoria$[i][1][1] \leftarrow d(s_0, v_i) + d(r_1, v_i) +$ Memoria$[i][2][1]$
end for
$s_1 \leftarrow \arg\min_{i\in\{1,\dots,n\}}$ Memoria$[i][1][1]$
for $i \leftarrow 1$ **to** n **do**
    Memoria$[i][1][2] \leftarrow s_1$
end for
Secuencia $\leftarrow [s_1]$
$v \leftarrow s_1$
for $j \leftarrow 2$ **to** k **do**
    $v \leftarrow$ Memoria$[v][j][2]$
    Secuencia.append(v)
end for
return $OPT(s_0 r_1 \dots r_k)$, Secuencia $s_1 \dots s_k$

Para ver su implementación revisar anexo A.

Ejemplo de la ejecución del Algoritmo 1

Para ilustrar el funcionamiento del algoritmo diseñado (Algoritmo 1), se presenta un ejemplo con un grafo simple no dirigido $G = (V, E)$ con la entrada $s_0 = v_3, r_1 = v_5, r_2 = v_3, r_3 = v_2, r_4 = v_2$ que se ilustra en la figura descrita a continuación:



El Algoritmo 1 usa la siguiente matriz de distancias del grafo G, en donde la entrada (i, j) corresponde a la distancia entre el vértice v_i y el vértice v_j en G.

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 & 3 \\ 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 3 & 1 & 2 & 0 \end{pmatrix}$$

En orden de ejecución, primero se crea una matriz de memoria tridimensional con $n = 5$ y $k = 4$, quedando $M_{5x4x2}(0)$. Luego, en el primer ciclo for, se calcula OPT$(v_i r_4) = d(v_i, v_2)$ y se agrega en la última columna junto con su vértice correspondiente, quedando una matriz de la siguiente forma:

$$\text{Memoria} = \begin{pmatrix} [0,0] & [0,0] & [0,0] & [1, v_1] \\ [0,0] & [0,0] & [0,0] & [0, v_2] \\ [0,0] & [0,0] & [0,0] & [2, v_3] \\ [0,0] & [0,0] & [0,0] & [1, v_4] \\ [0,0] & [0,0] & [0,0] & [3, v_5] \end{pmatrix}$$

A continuación, comienza el segundo procesamiento iterativo ($j = 3$ a $j = 2$). Para $j = 3$ ($r_3 = v_2$):

$$\text{Memoria}[i][3][1] = \min_{\ell}\{d(v_i, v_\ell) + d(v_2, v_\ell) + \text{Memoria}[i][4][1]\}$$

v_i	Cálculo	Memoria$[i][3]$
v_1	$\min\{0+1+1, 1+0+0, \ldots\} = 1$	[1, v_2]
v_2	$\min\{1+1+1, 0+0+0, \ldots\} = 0$	[0, v_2]
v_3	$\min\{1+1+1, 2+0+0, \ldots\} = 2$	[2, v_2]
v_4	$\min\{1+1+1, 1+0+0, \ldots\} = 1$	[1, v_2]
v_5	$\min\{2+1+1, 3+0+0, \ldots\} = 3$	[3, v_5]

Para $j = 2$ ($r_2 = v_3$):

$$\text{Memoria[i][2][1]} = \min_{\ell}\{d(v_i, v_\ell) + d(v_3, v_\ell) + \text{Memoria}[i][3][1]\}$$

v_i	Cálculo	Memoria$[i][2]$
v_1	$\text{mín}\{0+1+1, 1+1+0, \ldots\} = 2$	[2, v_1]
v_2	$\text{mín}\{1+1+1, 0+2+0, \ldots\} = 2$	[2, v_2]
v_3	$\text{mín}\{1+1+1, 2+0+0, \ldots\} = 2$	[2, v_3]
v_4	$\text{mín}\{1+1+1, 1+1+0, \ldots\} = 2$	[2, v_4]
v_5	$\text{mín}\{2+1+1, 3+1+0, \ldots\} = 3$	[3, v_3]

Finalmente, se procesa el primer requerimiento con la fuente inicial, usando $j = 1$ ($r_1 = v_5$ y $s_0 = v_3$):

$$\text{Memoria}[i][1][1] = d(v_i, v_5) + d(v_3, v_i) + \text{Memoria}[i][2][1]$$

v_i	Cálculo	Memoria$[i][1]$
v_1	$2+1+2=5$	[5, v_3]
v_2	$3+2+2=7$	[7, v_3]
v_3	$1+0+2=3$	[3, v_3]
v_4	$2+1+2=5$	[5, v_3]
v_5	$0+1+3=4$	[4, v_3]

Obteniendo la siguiente matriz de memoria:

$$\text{Memoria} = \begin{pmatrix} [5, v_3] & [2, v_1] & [1, v_2] & [1, v_1] \\ [7, v_3] & [2, v_2] & [0, v_2] & [0, v_2] \\ [3, v_3] & [2, v_3] & [2, v_2] & [2, v_3] \\ [5, v_3] & [2, v_4] & [1, v_2] & [1, v_4] \\ [4, v_3] & [3, v_3] & [3, v_2] & [3, v_5] \end{pmatrix}$$

Ahora bien, para la construcción de la secuencia óptima se busca el mínimo en $Memoria[\cdot][1][1]$, el cual corresponde v_3 con valor 3, para después buscar la siguiente fuente óptima en el 3er elemento de la columna siguiente. Siguiendo estas reglas se obtiene la siguiente secuencia reconstruida:

- $s_1 = v_3$ (de Memoria$[3][1][2]$)
- $s_2 = v_3$ (de Memoria$[3][2][2]$)
- $s_3 = v_2$ (de Memoria$[2][3][2]$)
- $s_4 = v_2$ (de Memoria$[2][4][2]$)

Así, se obtiene que el valor óptimo es 3, y su secuencia de fuentes óptima es $s_1 = v_3$, $s_2 = v_3$, $s_3 = v_2$, $s_4 = v_2$.

2.3. Correctitud y Tiempo de Ejecución

Una vez presentado el algoritmo que resuelve el PLD en su versión Off-line, Algoritmo 1, esta sección tiene como propósito establecer rigurosamente su correctitud y eficiencia. En primer lugar, en Teorema 5, se demuestra que el algoritmo efectivamente encuentra la secuencia óptima que minimiza la función objetivo, utilizando como base la estructura recursiva del problema formalizada en el Teorema 4. Posteriormente, se analiza su complejidad computacional, evaluando el costo temporal de cada componente del algoritmo. Ambos resultados constituyen fundamentos teóricos esenciales que validan tanto la correctitud como la viabilidad práctica del método propuesto.

Teorema 5. *El Algoritmo 1 (PLD Off-line) encuentra una secuencia óptima $s_1 s_2 \dots s_k$ que minimiza la función objetivo $\sum_{i=1}^{k} d(s_{i-1}, s_i) + d(r_i, s_i)$ para el Problema de Localización Dinámica Off-line.*

Demostración. La demostración procede por inducción hacia atrás sobre el número de requerimientos k, utilizando la estructura óptima de subproblemas expresada en el Teorema 4.

Para el último requerimiento r_k, el algoritmo inicializa correctamente:

$$\text{Memoria}[i][k][1] = \text{OPT}(v_i r_k) = d(v_i, r_k)$$

Esto es óptimo pues la única solución posible es $s_k = v_i$ con costo $d(v_i, r_k)$.

Se asume que para todo $j' > j$, Memoria$[i][j'][1]$ almacena correctamente $OPT(v_i r'_j \dots r_k)$. Para el paso j, el algoritmo calcula:

$$\text{Memoria}[i][j][1] = \min_{v_l \in V} \{d(v_i, v_l) + d(r_j, v_l) + \text{Memoria}[l][j+1][1]\}$$

Que corresponde exactamente a la relación de recurrencia del Teorema 4.

$$\text{OPT}(v_i r_j \dots r_k) = \min_{v \in V} \{d(v_i, v) + d(r_j, v) + \text{OPT}(v r_{j+1} \dots r_k)\}$$

Para $j = 1$, el algoritmo incorpora correctamente el nodo inicial s_0:

$$\text{Memoria}[i][1][1] = d(s_0, v_i) + d(r_1, v_i) + \text{Memoria}[i][2][1]$$

que implementa el caso particular del Teorema 4:

$$\text{OPT}(s_0 r_1 \dots r_k) = \min_{v \in V} \{d(s_0, v) + d(r_1, v) + \text{OPT}(v r_2 \dots r_k)\}$$

Por lo tanto, el algoritmo computa correctamente tanto el valor óptimo como la secuencia que lo alcanza, satisfaciendo completamente los requerimientos del problema. □

Para evaluar el desempeño del algoritmo, es necesario recurrir al análisis asintótico de su tiempo de ejecución. En este contexto, se emplea la notación $\mathcal{O}$-grande, denotada por $\mathcal{O}$, que permite expresar cotas superiores sobre funciones de complejidad, ignorando constantes y términos de menor orden. A continuación, se presenta su definición formal.

Definición 6. *Para una función dada $g(n)$, se denota por $\mathcal{O}(g(n))$ al conjunto de funciones*

$$\mathcal{O}(g(n)) = \{f(n) : \textit{existen constantes positivas } c \textit{ y } n_0 \textit{ tales que } 0 \leq f(n) \leq cg(n) \textit{ para todo } n \geq n_0\}$$

Se usa la notación $\mathcal{O}$ para dar una cota superior a una función, hasta un factor constante.

El siguiente teorema establece la complejidad temporal del algoritmo PLD Off-line en notación asintótica $\mathcal{O}$-grande. Notar que el Algoritmo PLD Off-line recibe como parte de la entrada la matriz de distancia de un grafo G, y su obtención tiene un tiempo de ejecución de entre $\mathcal{O}(n^2)$ y $\mathcal{O}(n^3)$, dependiendo de la cantidad de aristas del grafo G, en donde $n = |V(G)|$. En particular, se puede utilizar el algoritmo Dijkstra o BFS, pero para este estudio el más adecuado es el algoritmo BFS dada la simplicidad del grafo considerado [CLRS22].

Teorema 6. *El Algoritmo PLD Off-line tiene complejidad temporal $\mathcal{O}(n^2k)$, donde $n = |V(G)|$ y k es el número de requerimientos.*

Demostración. La complejidad se deriva del análisis de los cuatro componentes principales del algoritmo:

Primero, la inicialización de la estructura de memorización requiere un tiempo $\mathcal{O}(nk)$ para crear la matriz tridimensional y $\mathcal{O}(n)$ para poblar los valores base correspondientes al último requerimiento r_k. Este último paso se ejecuta mediante un único bucle sobre los n vértices, asignando en cada iteración el valor $d(v_i, r_k)$ en tiempo constante.

El núcleo del algoritmo reside en el llenado de la tabla de programación dinámica. Aquí, se anidan tres bucles: el más externo itera sobre los $k-2$ requerimientos restantes (desde $j = k-1$ hasta 2), el intermedio recorre los n vértices del grafo, y el interno calcula el mínimo sobre n posibles vértices v_l. Cada cálculo del mínimo evalúa en tiempo constante la expresión $d(v_i, v_l) + d(r_j, v_l) +$ Memoria$[l][j+1][1]$, asumiendo que las distancias están precomputadas en la matriz A. Por lo tanto, este bloque dominante consume $\mathcal{O}(k) \times \mathcal{O}(n) \times \mathcal{O}(n) = \mathcal{O}(n^2k)$ operaciones.

Posteriormente, el procesamiento del caso especial para $j = 1$ implica un único bucle sobre los n vértices, realizando operaciones constantes por iteración, lo que aporta un término $\mathcal{O}(n)$ adicional. Finalmente, la reconstrucción de la solución óptima requiere encontrar el mínimo inicial en $\mathcal{O}(n)$ tiempo y luego ejecutar $k-1$ pasos de seguimiento de punteros con costo constante cada uno, sumando $\mathcal{O}(n+k)$. □

Capítulo 3

Problema de Localización Dinámica On-line

Dada la definición de Windex de un grafo, no es claro a priori si todos los grafos tienen Windex finito. En efecto, como veremos en este capítulo, en el trabajo de Chung Graham y Saks [CGS87], se muestran una serie de grafos que no tienen Windex acotado y que por definición tienen Windex infinito. En un subsiguiente trabajo [CGS89], no solo muestran esto, si no que además caracterizan la clase de grafos con Windex finito. Este capítulo tiene por objetivo exhibir cotas y valores del Windex para clases de grafos específicas, comenzando por el estudio del Windex para una arista en Sección 3.1, en donde además se entrega una metodología para la obtención de cotas del Windex, para seguir en Sección 3.2 con dos ejemplos: un grafo con Windex infinito, y el Windex de los grafos completos. Recordemos que un grafo completo en n vértices, denotado por K_n, es un grafo con conjunto de aristas todos los pares de vértices. Para finalizar este capítulo, en Sección 3.3 presentamos y estudiamos un algoritmo que aproxima el PLD con radio 2 para cualquier grafo, y que trabaja dentro de una ventana 1.

3.1. Windex para una arista

Como ya hemos mencionado, se puede demostrar que un grafo tiene Windex al menos un valor, exactamente ese valor, o no acotado (infinito). Existen distintos enfoques para abordar las demostraciones en cada uno de estos casos, y vamos a revisar algunos de estos enfoques en esta sección.

Para probar que el Windex de un grafo G es exactamente un entero k, es decir que $WX(G) = k$, se debe demostrar que existe una cota inferior $WX(G) \geq k$ y una cota superior $WX(G) \leq k$. Para la cota superior, se debe mostrar que existe un algoritmo que tan solo teniendo la información de los k requerimientos siguientes, es capaz de encontrar una secuencia de posiciones de la fuente que sea óptima. En algunas ocasiones, por ejemplo, demostramos optimalidad del algoritmo a partir de una inducción en la secuencia de requerimientos (Proposición 10).

Para la corta inferior, se debe mostrar que para una ventana estrictamente menor que k, independientemente de la elección de la posición de la fuente que realize, existe un próximo requerimiento que hará que esta elección ya realizada no sea la óptima. Más concretamente, para probar que el Windex de un grafo G es más grande que cierto entero k, es decir $WX(G) \geq k$, se procede por contradicción suponiendo que $WX(G) \leq k-1$. Se define a conveniencia una instancia la cual está dada por una fuente inicial s_0 y una lista de $k-1$ requerimientos tal que para cada posible elección de s_1 desde s_0, existe un requerimiento r_k que refuta la optimalidad de la elección inicial s_1. Dicho de otra manera, si para una fuente inicial s_0 se tiene una secuencia de posiciones de la fuente entregada por el PLD $s_1 s_2 ... s_{k+1}$, existirá una secuencia distinta $s'_1 s'_2 ... s'_{k+1}$ de manera que

$$C(s_1 s_2 ... s_{k+1}; s_0 r_1 ... r_{k+1}) > C(s'_1 s'_2 ... s'_{k+1}; s_0 r_1 ... r_{k+1}).$$

Esta metodología se puede usar también para demostrar que $WX(G) = \infty$, suponiendo que G tiene Windex acotado por algún valor k, es decir $WX(G) \leq k$ y llegando a una contradicción.

Comenzamos estudiando el comportamiento del Windex desde el concepto más básico de un grafo conexo; esto es, un grafo con al menos una arista.

Proposición 7. *Para cualquier grafo G con al menos una arista, se cumple que $WX(G) \geq 2$.*

Demostración. Vamos a demostrar por contradicción. Sea G un grafo con al menos una arista uv, supongamos que $WX(G) = 1$ y que la fuente inicial será $s_0 = u$, entonces el siguiente requerimiento que se puede ver es $r_1 = v$. Para este caso existen dos posibles opciones: avanzar y actualizar la fuente a $s_1 = v$ o quedarme y actualizar la fuente a $s_1 = u$.
Si $s_1 = v$, la función objetivo aumentará de 0 a 1 (suponiendo que cada movimiento tiene costo 1). Ahora supongamos que la lista de los requerimientos siguientes es $uuuuu...$, entonces la fuente debe devolverse a $s_2 = u$ y mantenerse, aumentando la función objetivo a 2. Sin embargo, la decisión inicial de avanzar muestra que no fue una estrategia óptima, puesto que "quedarse" habría reducido el cálculo de la función objetivo en una unidad.
Si $s_1 = u$, la función objetivo aumentará de 0 a 1 (puesto que la distancia de u a v es 1). Ahora supongamos que la lista de los requerimientos siguientes es $vvvvv...$, entonces la fuente debe avanzar a $s_2 = v$ y mantenerse, aumentando la función objetivo a 2. Luego, se concluye lo mismo que el caso anterior.
Dado que se encontró una lista de requerimientos de tal manera que en ambos casos no fue óptimo el movimiento inicial, entonces $WX(G)$ no puede ser 1 dado que no existe una estrategia para obtener una secuencia óptima, por lo que se concluye que $WX(G) \geq 2$. □

Esta proposición muestra que cualquier grafo con al menos una arista tendrá Windex al menos 2, y para su demostración se utilizó un grafo de una arista, para luego concluir que un grafo más grande a este puede tener Windex mayor o igual a 2. Entonces, una pregunta natural sería determinar el Windex de este grafo G de una sola arista.

Proposición 8. *Para un grafo G con una arista, se cumple que $WX(G) = 2$.*

Demostración. Para esta demostración, se busca una cota inferior y una cota superior. Dada la proposición anterior, se tiene la cota inferior $WX(G) \geq 2$. Falta encontrar una cota superior de manera que $WX(G) \leq 2$, aplicando un algoritmo que entregue una solución óptima dentro de una ventana 2. Sea $s_0 r_1 r_2 \dots r_n$ una instancia del PLD en G. Vamos a definir una secuencia $s_1 s_2 \dots s_n$ óptima para el PLD en G tal que s_{i+1} se obtiene a partir de s_i, r_{i+1} y r_{i+2}.
Para esto, consideramos el siguiente algoritmo: sea s_i el estado actual, y sean r_{i+1}, r_{i+2} los siguientes 2 requerimientos. Sea r el primer vértice repetido en la secuencia $s_i r_{i+1} r_{i+2}$. Si el vértice repetido es $r = r_{i+1}$, se elige la fuente siguiente s_{i+1} como r_{i+1}, sino la fuente se mantiene $s_{i+1} = s_i$. En otras palabras, si el vértice repetido es igual al primer requerimiento entonces se actualiza la fuente, y si no es igual entonces la fuente se queda igual (Ver figura 3.1).

Caso base: sean r_1, r_2 los requerimientos del PLD con s_0 como la fuente inicial. Se tienen dos posibles casos para iniciar el algoritmo:

- Si $s_0 = r_1$, se tiene que $s_1 = s_0$. Luego, si $r_2 = r_1$ entonces $s_2 = s_1$ y el costo es de 0 total dado que no se realizó ningún cambio de fuente. Por otro lado, si $r_2 \neq r_1$, el algoritmo cambiará la fuente a $s_2 = r_2$ y el costo total es de 1 puesto que la fuente cambió solo una vez.

- Si $s_0 \neq r_1$, se tiene que $s_1 = r_1$. Luego, si $r_2 = r_1$ la fuente se mantiene y queda $s_2 = s_1$ y el costo total es de 1 dado que se realizó solo un cambio. Por otro lado, si $r_2 \neq r_1$, el algoritmo cambiará la fuente a $s_2 = r_2$ y el costo total es de 2 dado que se realizaron dos cambios.

Así, para cada caso posible se cumple que el costo calculado es el mínimo necesario que se puede obtener.

Supongamos cierta la siguiente hipótesis inductiva para n requerimientos: $P(n)$: para un Windex 2, el algoritmo mencionado garantiza un costo mínimo para una secuencia de n requerimientos $r_1 r_2 ... r_n$. Consideramos ahora una secuencia de $n+1$ requerimientos, $r_1 r_2 ... r_n r_{n+1}$. La hipótesis inductiva nos asegura que el algoritmo ya ha minimizado el costo para los primeros n requerimientos. Entonces,

- Si $r_{n+1} = r_n$: El algoritmo decide quedarse en el estado s_n, ya que el próximo vértice solicitado es el mismo que el actual. Así, no realiza ningún cambio de estado adicional, lo cual minimiza el costo.

- Si $r_{n+1} \neq r_n$: El algoritmo cambia el estado al vértice r_{n+1}, ya que es necesario para satisfacer la nueva solicitud. Este cambio de estado es inevitable para satisfacer r_{n+1}, y por lo tanto, también minimiza el costo.

Para ambos casos el algoritmo asegura mover la fuente sólo cuando es necesario para poder satisfacer el requerimiento $n+1$ solicitado, por lo que el algoritmo es óptimo para la secuencia de longitud $n+1$ requerimientos.

Esto prueba que $WX(G) = 2$, ya que una ventana de tamaño 2 es suficiente para asegurar el mínimo costo en cualquier secuencia de requerimientos. □

Para entender mejor el algoritmo usado para la demostración anterior, se expone el siguiente ejemplo en la Figura 3.1.

Figura 3.1: Para una fuente en posición inicial A, con requerimientos $r_1 = B, r_2 = A$ y $r_3 = B$, el algoritmo funciona de la siguiente manera: ABA tiene como vértice repetido $r = A$, y como $r_1 \neq A$ entonces $s_1 = A$. Ahora la secuencia es AAB y como $r_2 = A$ entonces $s_2 = B$. Ahora la secuencia puede ser ABA o ABB, pero en cualquiera de estos dos casos ya se sabe como seguir.

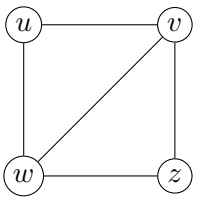
En la siguiente sección veremos dos ejemplos: un grafo con Windex infinito, y el Windex de los grafos completos.

3.2. Windex infinito y Windex de grafos completos

Hemos mencionado al principio que existen grafos para los cuales el Windex no es finito. De hecho, la mayoría de los grafos no tienen Windex finito (como lo muestran Chung, Graham y Saks [CGS89]). En la siguiente proposición, mostramos un ejemplo, tomado de [CGS87], que indica que el Windex del grafo completo con cuatro vértices menos una arista no es acotado. Otros ejemplos de grafos con Windex no acotado serían el $K_{2,3}$, es decir el grafo bipartito completo con biparticiones de dos y tres vértices, y el ciclo con cinco vértices C_5.

Proposición 9. *Sea $G = K_4 - e$ un grafo completo de 4 vértices sin una arista. Entonces $WX(G) = \infty$.*

Demostración. Se procede por contradicción suponiendo que existe un entero k tal que $WX(G) \leq k$. Sea $V(G) = \{u, v, w, z\}$ los vertices de G y sea uz la única arista que no pertenece a K_4.



Supongamos que la fuente inicial está en $s_0 = v$, y consideremos la siguiente lista finita de requerimientos:

$$\sigma = wuzuz \dots uz,$$

donde la subsecuencia de dos elementos uz se repite k veces.

Cada aparición de uz en σ tiene un costo de $d(u, z) = 2$, y como uz se repite k veces, el costo acumulado es $2k$. Además, se debe contar el primer movimiento hacia w, que agrega $d(w, u) = 1$, dando un costo total de al menos $2k+1$. Si el siguiente requerimiento después de la lista entregada es v, entonces el costo total en $2k + 1$ se mantiene cuando se tiene $s_1 = v$, por lo que $s_1 = v$ es óptimo. Si $s_1 \neq v$, suceden al menos dos cambios de posición de fuente, aumentando el costo al menos $2k+2$. Como k puede ser arbitrariamente grande, no existe un algoritmo puede determinar un estado óptimo dentro de una ventana finita, lo que contradice la suposición de que $WX(G)$ es finito. □

Puesto que para diferentes tipos de grafos, su Windex puede cambiar de manera no intuituva, entonces ahora se presenta el estudio del Windex de los grafos completos, el cual si es finito.

Proposición 10. *Para todo $n \geq 2$, un grafo completo de n vértices K_n cumple que $WX(K_n) = n$.*

Demostración. Se demostrará la igualdad buscando una cota superior y una cota inferior para el costo total n a partir de una secuencia de requerimientos entregada dentro de una ventana n.

Sea $V(K_n) = u_0, u_1, ..., u_{n-1}$ y se considera la siguiente secuencia de requerimientos:

$$\sigma = u_0 u_1 ... u_{n-1}$$

donde u_0 es el estado inicial. Si el primer requerimiento después de σ es u_0, entonces la secuencia óptima debe comenzar y continuar con u_0 porque así el costo es de $n - 1$, mientras que para cualquier otro caso será el costo igual a n. Análogamente, notamos que una secuencia de estados óptimos deben comenzar con u_1 si el primer requerimiento después de σ es u_1. Esto prueba que $WX(K_n) > n - 1$.

Para probar que $WX(K_n) \leq n$, consideramos el siguiente algoritmo: Sea s_i el estado actual, y sea $r_{i+1} r_{i+2} ... r_{i+n}$ los siguientes n requerimientos. Sea r el primer vértice repetido en la secuencia $s_i r_{i+1} r_{i+2} ... r_{i+n}$. Elegimos la fuente s_{i+1} como r_{i+1} si $r = r_{i+1}$, sino $s_{i+1} = s_i$.

Caso base: sean $r_1 r_2 ... r_n$ los requerimientos del DLP con s_0 como la fuente inicial. A partir de r_1, se determina el primer vértice repetido en la secuencia $r_1 r_2 ... r_n$. Se tienen dos posibles casos para iniciar el algoritmo:

- Si el primer vértice repetido es $r = r_1$, entonces elegimos como la siguiente fuente $s_1 = r_1$, aumentando el costo total en 1.
- si $r \neq r_1$ entonces la fuente se mantiene, manteniendo el costo mínimo dado que no se aumenta el costo total.

Así, para cada caso posible se cumple que el costo calculado es el mínimo necesario que se puede obtener.

Supongamos cierta la siguiente hipótesis inductiva para $k \geq n$ requerimientos: $P(k)$: para un Windex n, el algoritmo mencionado garantiza un costo mínimo para una secuencia de k requerimientos $r_1 r_2 ... r_k$. Consideramos ahora una secuencia de $k+1$ requerimientos, $r_1 r_2 ... r_k r_{k+1}$. La hipótesis inductiva nos asegura que el algoritmo ya ha minimizado el costo para los primeros k requerimientos. Entonces según el algoritmo proporcionado,

- Si $r_{k+1} = r_k$: El algoritmo decide quedarse en el estado s_k, ya que el próximo vértice solicitado es el mismo que el actual. Así, no realiza ningún cambio de estado adicional, lo cual minimiza el costo.
- Si $r_{k+1} \neq r_k$: El algoritmo cambia el estado al vértice r_{k+1}, ya que es necesario para satisfacer la nueva solicitud. Este cambio de estado es inevitable para satisfacer r_{k+1}, y por lo tanto, también minimiza el costo.

Para ambos casos el algoritmo asegura mover la fuente sólo cuando es necesario para poder satisfacer el requerimiento $k+1$ solicitado, por lo que el algoritmo es óptimo para la secuencia de longitud $k+1$ requerimientos.
Esto prueba que $WX(G) = n$, ya que una ventana de tamaño n es suficiente para asegurar el mínimo costo en cualquier secuencia de requerimientos. □

3.3. Algoritmo aproximado de ventana 1

Si bien el algoritmo presentado en el Capítulo 2 (Algoritmo 1) permite encontrar una solución óptima para el PLD en su versión Off-line, nos interesa estudiar algoritmos que trabajan dentro de ventanas acotadas. Como ya vimos en la Sección 3.2 no todos los grafos tienen Windex finito, por lo que un primer paso para el diseño de algoritmos On-line sería pedir que aproximen el PLD con un buen radio de aproximación. El propósito de esta sección es presentar un algoritmo aproximado con radio 2 que trabaja dentro de una ventana 1 y que sigue el paradigma de diseño glotón (o greedy, en inglés). Probaremos que este algoritmo sirve como prueba de la validez de la conjetura formulada en 1 para el caso $k = 1$ acerca de la existencia de algoritmos On-line con buenos radios de aproximación. Por otro lado, en el Capítulo 5 utilizaremos este algoritmo para estudiar empíricamente su desempeño, y veremos que dependiendo de las clases de grafos en donde estemos trabajando, este algoritmo puede ser mucho mejor que lo que se puede demostrar teóricamente.

Como dijimos anteriormente, el algoritmo glotón aquí propuesto cumple con la característica de trabajar con ventana 1, ya que decide su acción basándose únicamente en el estado actual y el requerimientos siguiente. Esta propiedad lo convierte en un candidato natural para explorar experimentalmente los límites del rendimiento On-Line bajo información parcial. La idea central del algoritmo es seleccionar en cada paso la ubicación que minimiza localmente el costo de atención a los requerimientos inmediatos, sin considerar el efecto futuro de dicha elección.

A continuación, se expone el pseudocódigo del algoritmo.

Algoritmo 2 Algoritmo Glotón PLD

Entrada: Matriz de distancias A para $G = (V, E)$, fuente inicial s_0, secuencia $r_1 r_2 \dots r_k$
Salida: Secuencia $s_1 s_2 \dots s_k$ y costo total $\min \sum_{i=1}^{k} d(s_{i-1}, s_i) + d(s_i, r_i)$

1: $costo \leftarrow 0$
2: $S \leftarrow [\,]$
3: **for** $i \leftarrow 1$ **to** k **do**
4: $\quad s_i \leftarrow r_i$
5: $\quad$ Agregar s_i a S
6: $\quad costo \leftarrow costo + d(s_{i-1}, s_i)$
7: **end for**
8: **return** Secuencia S, costo total $costo$

Para ver su implementación revisar anexo A.

Notar que el Algoritmo 2 Glotón recibe como parte de la entrada la matriz de distancia de un grafo G, con la finalidad de no interferir con los tiempos de ejecución calculados para cada algoritmo propuesto en este estudio.

El uso del Algoritmo 2 (Glotón PLD) no solo se justifica por su eficiencia, teniendo un tiempo de ejecución $O(k)$ recibiendo como entrada la matriz de distancias par el grafo G con n vértices y la secuencia de requerimientos $s_0 r_1 r_2 \dots r_k$, sino también por el siguiente resultado teórico que respalda su desempeño. En particular, la siguiente proposición muestra que el resultado del costo total entregado por este algoritmo no se alejan demasiado del costo total óptimo.

Proposición 11. *Sea $\mathcal{A}$ el Algoritmo 2. Entonces, para cualquier grafo G, el algoritmo $\mathcal{A}$ aproxima el PLD en G con radio 2. Además, el Algoritmo $\mathcal{A}$ trabaja dentro de una ventana 1.*

Demostración. Consideramos la instancia $s_0 r_1 r_2 \dots r_k$ del PLD en G. Dado que el algoritmo glotón PLD define $s_i = r_i$ para todo $i \geq 1$, el costo total se puede escribir como:

$$\begin{aligned} C_{\mathcal{A}}(s_0 r_1 r_2 \dots r_k) &= \sum_{i=1}^{k} d(s_{i-1}, s_i) + d(s_i, r_i) \\ &= d(s_0, r_1) + \sum_{i=2}^{k} d(r_{i-1}, r_i), \end{aligned}$$

Para acotar este costo, se considera una solución óptima $s'_1 \dots s'_k$ y aplicamos la desigualdad triangular:

$$d(r_{i-1}, r_i) \leq d(r_{i-1}, s'_{i-1}) + d(s'_{i-1}, s'_i) + d(s'_i, r_i).$$

Sumando para $i = 2$ hasta k y agregando $d(s_0, r_1) \leq d(s_0, s'_1) + d(s'_1, r_1)$ por desigualdad

triangular, se obtiene:

$$\begin{aligned} C_{\mathcal{A}}(s_0 r_1 r_2 \dots r_k) &\leq d(s_0, s'_1) + d(s'_1, r_1) + \sum_{i=2}^{k} d(r_{i-1}, s'_{i-1}) + d(s'_{i-1}, s'_i) + d(s'_i, r_i) \\ &\leq \sum_{i=1}^{k} d(s'_{i-1}, s'_i) + 2\sum_{i=1}^{k} d(s'_i, r_i) \\ &\leq 2 \cdot OPT(s_0 r_1 r_2 \dots r_k) \end{aligned}$$

□

Capítulo 4

Grafos con Window Index dos

En este capítulo nos centraremos en el estudio de grafos con Windex dos. Chung, Graham y Saks [CGS87] caracterizaron los grafos con Windex dos, y exhibieron un algoritmo óptimo para todos aquellos grafos con Windex 2. En la Sección 4.1, vamos a introducir el concepto de grafo mediano, a través de la Propiedad de la Tripleta de Steiner Única (USTP, por sus siglas en inglés), y vamos a demostrar que los árboles son un caso particular de grafos medianos. En la Sección 4.2, enunciamos el resultado de Chung, Graham y Saks [CGS87] acerca de la caracterización de los grafos con windex dos, probamos que los árboles tienen windex 2, que el windex se comporta muy bien bajo el producto cartesiano de grafos, y que los grafos de windex dos son grafos medianos. En esta misma sección, introducimos la estrategia USTP, la cual da lugar a un algoritmo aproximado que trabaja dentro de una ventana dos, y que es óptima para los grafos de windex 2. Para finalizar, en Sección 4.3 presentamos el algoritmo en base a la estrategia USTP, y argumentamos su correctitud para grafos medianos, y su tiempo de ejecución.

4.1. Grafos medianos

Para comenzar, se presentan una serie de definiciones extraídas de [CGS89], que servirán como base para el desarrollo posterior. En todas las definiciones que siguen se considerarán tres vértices $a, b, c \in V(G)$ no necesariamente distintos. Esta convención resulta más conveniente para las demostraciones y no altera la validez de las definiciones.

La primera definición que será introducida corresponde a Steiner Point, la cual identifica el vértice que minimiza la suma de distancias entre tres vértices a considerar.

Definición 7 (Steiner Point)**.** *Sea G un grafo. Un vértice $v \in V(G)$ se denomina* Steiner Point *para la tripleta de vértices $a, b, c \in V(G)$ si satisface la siguiente desigualdad:*

$$d(v, a) + d(v, b) + d(v, c) \leq d(x, a) + d(x, b) + d(x, c)$$

para todo $x \in V(G)$. El conjunto de todos los Steiner points de la tripleta a, b, c se denota por $S(a, b, c)$.

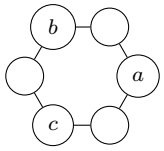
A partir de esta noción, surge una propiedad estructural relevante para ciertos tipos de grafos. Se pide que para cualquier tripleta de vértices exista exactamente un único Steiner Point. Esto conduce a la siguiente definición.

Definición 8 (Propiedad USTP)**.** *Sea G un grafo. Diremos que G satisface la Propiedad de la Tripleta Steiner Única (USTP en inglés) si para cualesquiera tres vértices $a, b, c \in V(G)$ se tiene que $|S(a, b, c)| = 1$.*

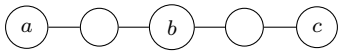
La propiedad USTP no solo describe una característica particular sino que también es equivalente a la definición de grafo mediano, brindando una interpretación útil en términos de caminos más cortos.

Definición 9 (Grafos Medianos)**.** *Un grafo G es mediano si y solo si satisface la propiedad USTP. De forma equivalente, un grafo G es mediano si y solo si existe un único vértice en la intersección de los caminos más cortos entre a y b, a y c, y b y c (y este vértice es un Steiner Point), para todo $a, b, c \in V(G)$.*

Para entender mejor estos conceptos se presentan los siguientes ejemplos en la Figura 4.1 correspondientes a un ciclo de par de 6 vértices y un camino de 5 vértices, en donde el ciclo no es un grafo mediano, pero el camino sí lo es.



(a) En este ciclo, para la tripleta a, b, c se tiene que existen 3 vértices (distintos a la tripleta mencionada) que minimizan la suma de distancias exigida por definición de grafos medianos, de modo que no se cumple la unicidad de Steiner Point.



(b) En este camino, para la tripleta a, b, c, el vértice b es el único que minimiza la suma de distancias exigida por definición de grafo mediano y también pertenece a la intersección de los tres caminos más cortos, cumpliéndose la unicidad de Steiner Point.

Figura 4.1

Un caso de especial interés es el de los árboles dada su estructura acíclica y de caminos únicos entre cada par de vértices, lo cual sugiere la existencia de un vértice que podría garantizar la propiedad USTP. El siguiente lema formaliza esta observación.

Lema 12. *Si G es un árbol, entonces G es un grafo mediano.*

Demostración. Probaremos que G satisface la propiedad USTP. Sean x, y, z distintos vértices en G. Si x, y, z pertenecen a un camino P del árbol, entonces podemos asumir sin pérdida de generalidad que x y z son los extremos de P, y el vértice y es un vértice interno de P, ver Figura 4.2.

Dado que existe un único camino en G que conecta a x y z, este debe ser P, y por lo tanto y es un Steiner point de x, y, z, y en efecto, el único. Ahora podemos suponer que x, y, z no pertenecen

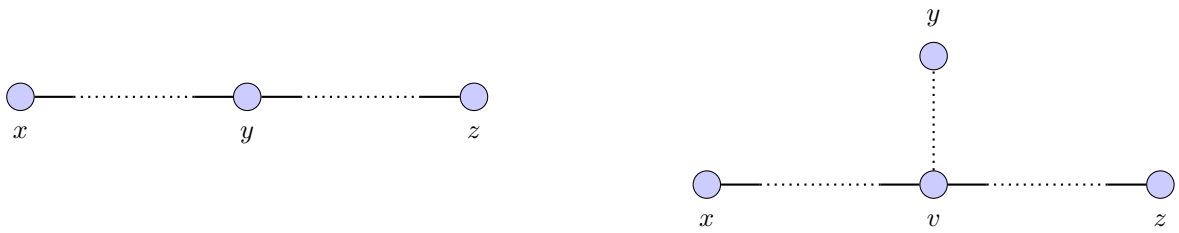


Figura 4.2: A la izquierda el primer caso de la demostración del Lema 12 en donde x, y, z están contenidos en un camino del árbol. A la derecha el caso contrario, cuando x, y, z no están contenidos en un camino del árbol.

a un camino del árbol. Entonces, como G es un árbol, existe un único camino entre x e y, digamos P_{xy}, lo mismo para x y z, digamos P_{xz} e igualmente para y y z, digamos P_{yz}, y la intersección entre ellos $P_{xy} \cap Pxz \cap P_{zy}$ corresponde a un único vértice v en G, el cual por consecuencia es un Steiner point y es único. □

Se hace mención de otros ejemplos de clases de grafos medianos en [CGS89] los cuales corresponden a grillas planares e hypercubos. En particular, se menciona en el Corolario 16 que estos grafos tienen windex 2, y dado el Lema 18 se concluye que son grafos medianos.

4.2. Caracterización de grafos con Windex 2

Las siguientes propiedades de las secuencias óptimas para el PLD son sencillas pero muy útiles en las demostraciones que vendrán a continuación, por completitud se harán sus demostraciones. Informalmente, la primera propiedad dice que cada posición de la fuente minimiza la distancia entre la posición anterior, el requerimiento actual y la siguiente posición de la fuente. La segunda propiedad asegura que en los grafos medianos, cada posición de la fuente se encuentra en un camino más corto entre la posicióm anterior y el requerimiento actual.

Proposición 13. *Sea G un grafo, $s_0 r_1 r_2 \dots r_n$ una instancia del PLD en G. Suponga que $s_1 s_2 \dots s_n$ es una secuencia óptima para la instancia $s_0 r_1 r_2 \dots r_n$.*

(i) Para cada $i \in \{1, ..., n-1\}$, s_i es un Steiner Point de s_{i-1}, r_i, s_{i+1}.

(ii) Si G es un grafo mediano, entonces para cada $i \in \{1, ..., n-1\}$, el vértice s_i está en un camino más corto entre s_{i-1} y r_i.

Demostración.

(i) Es claro que el costo para el PLD viene dada la formulación

$$\sum_{i=1}^{n} d(s_{i-1}, s_i) + d(s_i, r_i) = \min_{s'_1 s'_2 \dots s'_n \in V(G)} \{\sum_{i=1}^{n} d(s'_{i-1}, s'_i) + d(s'_i, r_i)\}$$

Para un j cualquiera, sea $j \in \{1, ..., n\}$. Se demostrará que s_j es un Steiner Point de s_{j-1}, r_j, s_{j+1}, es decir:

$$d(s_j, s_{j-1}) + d(s_j, r_j) + d(s_j, s_{j+1}) = \min_{x \in V(G)} \{d(x, s_{j-1}) + d(x, r_j) + d(x, s_{j+1})\}$$

Por contradicción, se supondrá que s_j no es Steiner Point de s_{j-1}, r_j, s_{j+1}, por lo tanto existe algún vértice $x \in V(G)$ de manera que

$$d(s_j, s_{j-1}) + d(s_j, r_j) + d(s_j, s_{j+1}) > d(x, s_{j-1}) + d(x, r_j) + d(x, s_{j+1})$$

Luego, el costo para resolver los requerimientos $s_0 r_1 ... r_k$ a través de la secuencia óptima $s_1 s_2 ... s_n$ para el PLD viene dado por

$$\begin{aligned}
\sum_{i=1}^{n} d(s_{i-1}, s_i) + d(s_i, r_i) = &\sum_{i=1}^{j-1} d(s_{i-1}, s_i) + d(s_i, r_i) + d(s_j, s_{j-1}) + d(s_j, r_j) + d(s_j, s_{j+1}) \\
&+ d(s_{j+1}, r_{j+1}) + \sum_{i=j+2}^{n} d(s_{i-1}, s_i) + d(s_i, r_i) \\
&> \sum_{i=1}^{j-1} d(s_{i-1}, s_i) + d(s_i, r_i) + d(x, s_{j-1}) + d(x, r_j) + d(x, s_{j+1}) \\
&+ d(s_{j+1}, r_{j+1}) + \sum_{i=j+2}^{n} d(s_{i-1}, s_i) + d(s_i, r_i)
\end{aligned}$$

Esto indica que existe una contradicción en la minimalidad de la función óptima y por lo tanto, $s_1 s_2 ... s_n$ no sería secuencia óptima.

(ii) Por (i), y por definición de grafo mediano, tenemos que s_i es el único Steiner point para s_{i-1}, r_i, s_{i+1}. Y por lo tanto, en particular, s_i está en un camino más corto entre s_{i-1} y r_i.

□

Hasta ahora se ha establecido en el Lema 12 que todo árbol es un grafo mediano, por lo que resulta natural preguntarse si también se cumple que su Window Index es igual a 2. El siguiente teorema confirma esta afirmación, proporcionando así una caracterización directa del valor del Windex para este tipo de grafos.

Teorema 14. *Si G un árbol, entonces $WX(G) = 2$.*

Demostración. Sea G un grafo y $(s_0, r_1, r_2, \dots r_n)$ una instancia del PLD en G. Hemos probado, en el Lema 12, que los árboles satisfacen la propiedad USTP, y por lo tanto para cada tres vértices, existe un único Steiner point. Vamos a probar que la secuencia $s_1, \dots, s_n$ que se obtiene calculando inicialmente $s_1 = S(s_0, r_1, r_2)$, y luego $s_i = S(s_{i-1}, r_i, r_{i+1})$ para cada $i \in [n]$, es un secuencia óptima. Para esto, por Proposición 13(i), basta probar que la secuencia definida, satisface que $s_i = S(s_{i-1}, r_i, s_{i+1})$. Para esto, vamos a analizar dos casos de acuerdo a la configuración de s_{i-1}, r_i, r_{i+1} cuando estos son distintos:

Caso 1: Los vértices s_{i-1}, r_i, r_{i+1} forman un camino, llamémosle P. Vamos a mirar primero el caso en que los tres vértices s_{i-1}, r_i, r_{i+1} son distintos. En este caso, como s_i es steiner point, entonces es el único vértice en $\{s_{i-1}, r_i, r_{i+1}\}$ que queda al interior del camino P. Ver Figura 4.3.

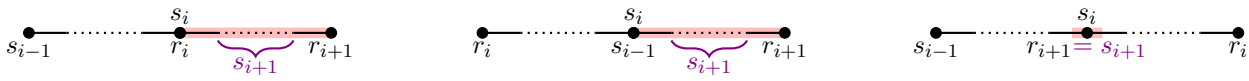


Figura 4.3: Configuraciones estructurales entre s_i, s_{i-1}, r_i y r_{i+1}

Como $s_{i+1} = S(s_i, r_{i+1}, r_{i+2})$, por Proposición 13(ii), s_{i+1} se encuentra en el camino que conecta s_i con r_{i+1}. Luego, si r_{i+1} es uno de los extremos del camino P, entonces s_{i+1} es un vértice que se encuentra en el camino P entre s_i y r_{i+1}, y por lo tanto, $s_i = S(s_{i-1}, r_i, s_{i+1})$. Ahora, si r_{i+1} no es un extremo del camino P, entonces $r_{i+1} = s_i$, y $s_{i+1} = S(s_i, r_{i+1}, r_{i+2}) = s_i$, y entonces $s_i = S(s_{i-1}, r_i, s_{i+1})$.

Caso 2: Los vértices s_{i-1}, r_i, r_{i+1} están conectados por caminos disjuntos a un vértices común v, el cual es su steiner point y que por lo tanto corresponde a s_i. Ver Figura 4.4.

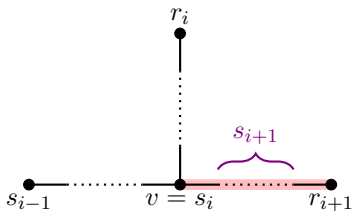


Figura 4.4: Caso 2 de la demostración del Teorema 14

Dado que, por Proposición 13(ii), s_{i+1} se encuentra en el camino entre r_{i+1} y s_i, se tiene que $s_i = S(s_{i-1}, r_i, s_{i+1})$.

Para finalizar, estudiamos el caso en que dentro de los tres vértices hay alguno repetido. En caso que los tres vértices son iguales, es trivial que $s_i = S(s_i, r_{i+1}, s_{i+1})$. Si hay dos repetidos, estos forman un camino P, en donde hay exactamente un vértice de s_{i-1}, r_i, r_{i+1} en un extremo, digamos x, y el otro extremo es por consecuencia s_i. Si $x = r_{i+1}$, entonces s_{i+1} se encuentra en el camino P entre r_{i+1} y s_i, y por lo tanto $s_i = S(s_{i-1}, r_i, s_{i+1})$. Si $x \neq r_{i+1}$, entonces r_{i+1} es igual a s_i, y por lo tanto $s_{i+1} = s_i$, y en consecuencia $s_i = S(s_{i-1}, r_i, s_{i+1})$.

□

La proposición a continuación es poderosa, dado que permite obtener una gran familia de grafos con Windex 2, a partir de la demostración que se ha hecho para los árboles. Para esto se procede a definir el producto cartesiano, producto el cual es vitalmente necesario para entender la proposición.

Definición 10 (Producto Cartesiano)**.** *Sea $G \square H$ el producto cartesiano de los grafos G y H. Por definición, los vértices de $G \square H$ son los pares (u, v) con $u \in V(G)$ y $v \in V(H)$, y existe una arista*

entre (u, v) *y* (u', v') *si y sólo si:*

$$\big(u = u' \text{ y } vv' \in E(H)\big) \quad \text{o} \quad \big(v = v' \text{ y } uu' \in E(G)\big).$$

Esto implica que la distancia entre dos vértices en $G \square H$ *se puede descomponer como:*

$$d_{G\square H}\big((u, v), (u', v')\big) = d_G(u, u') + d_H(v, v').$$

Con esta definición, ahora es posible establecer una relación directa entre el Window Index de dos grafos (no necesariamente distintos) y el Window Index de su producto cartesiano. El siguiente resultado muestra que el valor de WX para $G \square H$ está determinado únicamente por el mayor de los valores de WX de G y H, lo que permite extender de forma inmediata la propiedad $WX = 2$ a una amplia clase de grafos construidos mediante este producto.

Proposición 15. *Sean* G *y* H *grafos. Entonces*

$$WX(G\square H) = max\{WX(G), WX(H)\}.$$

Demostración. Sea $(s_0, s'_0)(r_1, r'_1)\dots(r_k, r'_k)$ una secuencia de requerimientos en $G \square H$. Supongamos que $s_1 s_2 \dots s_k$ es una secuencia óptima para $r_1 r_2 \dots r_k$ en G, y que $s'_1 s'_2 \dots s'_k$ es una secuencia óptima para $r'_1 r'_2 \dots r'_k$ en H.

Por definición de producto cartesiano, se tiene entonces que $(s_1, s'_1)(s_2, s'_2)\dots(s_k, s'_k)$ es la secuencia óptima para los requerimientos en $G \square H$. Se considera por simplicidad que $C_{G\square H} = C_{G\square H}((s_1, s'_1)\dots(s_k, s'_k); (s_0, s'_0)(r_1, r'_1)\dots(r_k, r'_k))$. Entonces su costo total es:

$$\begin{aligned} C_{G\square H} &= \sum_{i=1}^{k} d_{G\square H}\big((s_{i-1}, s'_{i-1}), (s_i, s'_i)\big) + d_{G\square H}\big((s_i, s'_i), (r_i, r'_i)\big) \\ &= \sum_{i=1}^{k} d_G(s_{i-1}, s_i) + d_H(s'_{i-1}, s'_i) + d_G(s_i, r_i) + d_H(s'_i, r'_i) \\ &= \sum_{i=1}^{k} [d_G(s_{i-1}, s_i) + d_G(s_i, r_i)] + \big[d_H(s'_{i-1}, s'_i) + d_H(s'_i, r'_i)\big]. \end{aligned}$$

Por lo tanto, el costo de la secuencia óptima en $G\square H$ conlleva que

$$WX(G\square H) \leq \text{máx}\{WX(G), WX(H)\}$$

Finalmente, cualquier otra secuencia de posiciones de fuentes en $G\square H$ no puede tener un costo menor. Por lo tanto,

$$WX(G\square H) \geq \text{máx}\{WX(G), WX(H)\}.$$

Se concluye que:

$$WX(G\square H) = \text{máx}\{WX(G), WX(H)\}.$$

□

Usando el Teorema 14 y la Proposición 15, se puede obtener por ejemplo que las grillas cuadradas y los grafos hipercubos tienen windex 2. Esto, dado que las grillas cuadradas son producto cartesiano de caminos, y los hipercubos son producto cartesiano de k_2, es decir, un grafo de 2 vértices y 1 arista.

Corolario 16. *Las grillas cuadradas y los hipercubos tienen windex 2.*

En [CGS89], Chung Graham y Saks, demuestran que los grafos con windex dos son exactamente los grafos medianos, y que además, la estrategia USTP es óptima, estrategia que consiste en elegir la posición de la fuente s_i según el vértice del grafo que minimice la suma desde s_i a los tres vértices s_{i-1}, r_i y r_{i+1}. En la Sección 4.3 se describe el pseudocódigo del algoritmo USTP y se calcula su tiempo de ejecución.

Teorema 17. *Sea G un grafo. Tenemos que $WX(G) = 2$ si y solo sí G satisface la propiedad USTP. Más aún, si G es un grafo que satisface la propiedad USTP, y $s_0 r_1 r_2 \dots r_n)$ es una instancia del PLD en G, entonces la secuencia $s_1 s_2 \dots s_n$ en donde $s_i = S(s_{i-1}, r_i, r_{i+1})$ es una secuencia óptima para la instancia $s_0 r_1 r_2 \dots r_n$.*

La demostración del Teorema 17 sigue de los dos lemas que vienen a continuación. El Lema 18 es probado en [CGS87]. Por completitud se incluye la demostración en este trabajo. Para la demostración es necesario introducir dos definiciones.

Definición 11 (Caterpillar)**.** *Un* Caterpillar *es un árbol compuesto por un camino principal formado por los vértices internos $s_1, s_2, \dots, s_k$, al cual se conectan hojas $s_0, r_1, r_2, \dots, r_k$. Esta estructura se asemeja a una oruga, donde las hojas emergen del camino principal. Ver Figura 4.5*

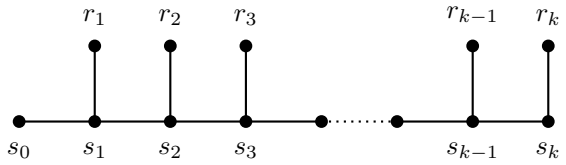


Figura 4.5

Definición 12 (Caterpillar Minimal)**.** *Un Caterpillar Minimal es un Caterpillar el cual cada vértice interno s_i con $1 \leq i < k$ es un* Steiner Point *para el conjunto $\{s_{i-1}, r_i, s_{i+1}\}$.*

Lema 18. *Si G es un grafo que satisface $WX(G) = 2$, entonces G satisface la propiedad USTP.*

Demostración. Supongamos que $WX(G) = 2$. Entonces existe un algoritmo A que resuelve el PLD en G con ventana 2, es decir, A entrega una secuencia óptima de fuentes de manera que la fuente s_i se determina a partir de s_{i-1}, r_i y r_{i+1}. Ahora bien, sea a, b, c una tripleta de vértices distintos en G que tienen dos Steiner Point, s y s'. Por definición de Steiner Point,

$$d(a, s) + d(b, s) + d(c, s) = d(a, s') + d(b, s') + d(c, s') = t.$$

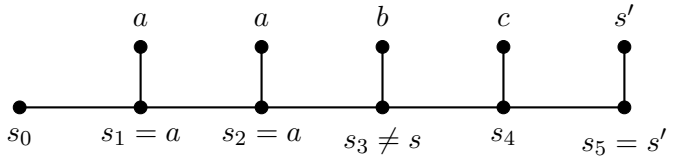


Figura 4.6

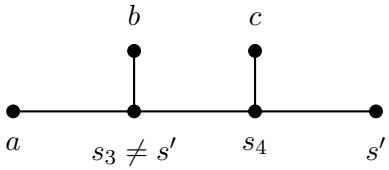


Figura 4.7

Se elegirán a, b, c de manera que t sea mínimo. Se quiere demostrar unicidad de Steiner Point. Para esto, se consideran subsecuencias convenientes de requerimientos de manera que al aplicar el algoritmo de windex 2 sobre cada uno se llegue a una contradicción.

Sea $aabczz$ la subsecuencia donde $r_1 = r_2 = a$, $r_3 = b$, $r_4 = c$, $r_5 = r_6 = z$ y donde z será un Steiner Point, s o s'. Sea $s_0 s_1 s_2 \ldots$ la secuencia de fuentes óptima obtenida por el algoritmo A. Es claro que $s_1 = s_2 = a$ y el algoritmo muestra bc. Dado que los últimos dos requerimientos a resolver son z, entonces es claro también que $s_5 = s_6 = z$. Sin pérdida de generalidad, se asume $s_3 \neq s'$ (dado z conveniente) y el algoritmo muestra cz, con $z = s'$. Luego, $s_5 = s_6 = s'$.

Tenemos el siguiente caterpillar en la Figura 4.6 y su minimal caterpillar corresponde a la Figura 4.7.

Para determinar s_4, se demostrará que

$$d(a, s_3) + d(b, s_3) + d(s_3, s_4) + d(s_4, c) + d(s_4, s') = t$$

Por definición de optimalidad de PLD, se debe cumplir que

$$d(a, s_3) + d(b, s_3) + d(s_3, s_4) + d(s_4, c) + d(s_4, s') = \min_{x,y}\{d(a, x) + d(b, x) + d(x, y) + d(y, c) + d(y, s')\}$$

Se mostrará que $\min_{x,y} d(a, x) + d(b, x) + d(x, y) + d(y, c) + d(y, s') = t$.

Comenzando por la desigualdad $\leq$. Si $x = y = s'$ se tiene que $d(a, s') + d(b, s') + d(c, s') = t$ y por lo tanto,

$$\min_{x,y} d(a, x) + d(b, x) + d(x, y) + d(y, c) + d(y, s') \leq t$$

Ahora la desigualdad $\geq$. $\forall x, y \in V(G)$,

$$\begin{aligned} d(a, x) + d(b, x) + d(x, y) + d(y, c) + d(y, s') &\geq d(a, x) + d(b, x) + d(x, c) + d(y, s') \\ &\geq d(a, x) + d(b, x) + d(x, c) \end{aligned}$$

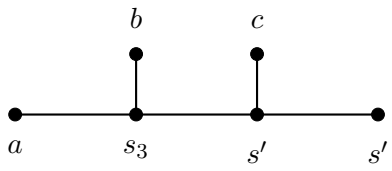


Figura 4.8

Por definición de Steiner Point para a, b, c:

$$d(a,x) + d(b,x) + d(x,c) \geq t \quad \Rightarrow \quad \min_{x,y}\{d(a,x) + d(b,x) + d(x,y) + d(y,c) + d(y,s')\} \geq t$$

Así,

$$\min_{x,y}\{d(a,x) + d(b,x) + d(x,y) + d(y,c) + d(y,s')\} = t$$

Luego, por minimalidad de t:

$$\begin{aligned} d(a,s_3) + d(b,s_3) + d(s_3,s_4) + d(s_4,c) + d(s_4,s') &\geq d(a,s_3) + d(b,s_3) + d(s_3,c) + d(s_4,s') \\ &\geq t + d(s_4,s') \end{aligned}$$

Entonces $t \geq t + d(s_4, s')$ y por minimalidad $d(s_4, s') = 0$. Así, $s_4 = s'$.

Ahora, dado que s' es un Steiner Point de a, b, c, entonces su caterpillar minimal es de largo t. Puesto que se estableció anteriormente que $s_3 \neq s'$, se demostrará que s_3 es un Steiner Point de a, b, c.

Sin pérdida de generalidad, $s_3 \neq s'$ y el caterpillar minimal tiene la Figura 4.8

Su largo es $d(a, s_3) + d(s_3, b) + d(s_3, s') + d(s', c)$ y por desigualdad triangular:

$$d(a,s_3) + d(s_3,b) + d(s_3,s') + d(s',c) \geq d(a,s_3) + d(s_3,b) + d(s_3,c)$$

Es claro que para todo vértice de G, la relación anterior será mayor que t dada la minimalidad de Steiner Point. Entonces:

$$d(a,s_3) + d(s_3,b) + d(s_3,c) \geq t \quad \forall s_3 \in V(G)$$

Como $d(a, s_3) + d(b, s_3) + d(s', s_3) + d(s', c) = t$, entonces $d(a, s_3) + d(b, s_3) + d(c, s_3) = t$, y s_3 es Steiner Point de a, b, c. Como $s_3 \neq s'$, y s_3 es Steiner Point, se puede suponer sin pérdida de generalidad que $s_3 = s$. Así, se afirma que:

$$d(a,s) + d(b,s) + d(s,s') + d(s',c) = t = d(a,s) + d(b,s) + d(c,s)$$

y luego $d(s, s') + d(s', c) = d(c, s)$ si s' está en un camino más corto entre s y c.

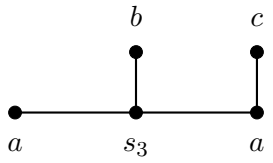


Figura 4.9

Por minimalidad de la tripleta a, b, c, se demostrará que $c = s'$ al suponer que $c \neq s'$, y que se está trabajando con la tripleta a, b, s'.

Puesto que la tripleta a, b, s' tiene al menos 2 Steiner Point, y dada la definición de Steiner Point y su minimalidad se puede asumir que

$$d(a, s) + d(b, s) + d(s', s) < t$$

De antemano se sabe que

$$d(a, s) + d(b, s) + d(c, s) = t = d(a, s') + d(b, s') + d(c, s')$$

y

$$d(a, s) + d(b, s) + d(s, s') + d(s', c) = t$$

Se tiene que

$$d(a, s) + d(b, s) + d(s', s) = t - d(s', c) < t \quad \text{si } d(s', c) > 0$$

y

$$d(a, s') + d(b, s') + d(s', s') = t - d(c, s') < t \quad \text{si } d(s', c) > 0$$

Entonces, si $d(s', c) > 0$, se contradice la minimalidad de la elección de $\{a, b, c\}$. Así, $d(s', c) = 0$ y se obtiene $c = s'$. Entonces, c es Steiner Point de $\{a, b, c\}$. A partir de las secuencias convenientes $ccbazz$ y $aacbzz$, se tiene que su caterpillar minimal tiene largo t y por simetría se puede azumir que a y b también son Steiner Point de $\{a, b, c\}$.

Dado que z debe ser un Steiner Point, se hace la suposición ahora que para la subsecuencia $aabczz$, $z = a$. Es claro que $s_1 = s_2 = a$ y por el procedimiento anterior, $s_6 = s_5 = s_4 = a$.

Su caterpillar minimal viene dado por la figura 4.9

Por minimalidad de la tripleta $\{a, b, c\}$ entonces $s_3 = a$.

Ahora, si $z = b$ entonces el mismo procedimiento fuerza que $s_1 = s_2 = a$ y $s_6 = s_5 = s_4 = s_3 = b$.

Finalmente, si z es cualquier Steiner Point y tiene largo t, por optimalidad de PLD se debe entregar la misma secuencia de fuentes óptimas, entonces a debe ser igual a b. Puesto que $a \neq b$, se llega a una contradicción.

□

La demostración del siguiente lema está en el Teorema 1 de [CGS87], utilizando una caracterización de grafos medianos en términos de conceptos topológicos, los cuales no formaron parte de estos estudios. Sin embargo, también se puede probar el Lema 19 refinando la demostración original desarrollada en esta tesis para los árboles (Teorema 14).

Lema 19. *Si G es un grafo que satisface la propiedad USTP, y $s_0 r_1 r_2 \dots r_n$ es una instancia del PLD en G, entonces la secuencia $s_1 s_2 \dots s_n$ en donde $s_i = S(s_{i-1}, r_i, r_{i+1})$ es una secuencia óptima para la instancia $s_0 r_1 r_2 \dots r_n$.*

4.3. Algoritmo aproximado con ventana 2

En esta sección se presentan dos algoritmos clave para realizar pruebas en el siguiente capítulo, además de su demostración de correctitud y tiempo de ejecución. En el Algoritmo 3 encontramos un subprocedimiento que, dado el estado previo y los dos próximos requerimientos, devuelve un vértice que minimiza la suma de tres distancias, que en el caso de los grafos medianos, es único. Su tiempo de ejecución corresponde a $\mathcal{O}(n)$

Algoritmo 3 Vértice minimizador

Entrada: A Matriz de distancias de $G = (V, E)$, s_{i-1}, r_i, r_{i+1}
Salida: Vértice que minimiza $d(s_{i-1}, v) + d(r_i, v) + d(r_{i+1}, v)$

```
1: costo ← ∞
2: vertice ← ∅
3: for v ∈ V(G) do
4:     suma ← d(s_{i-1}, v) + d(r_i, v) + d(r_{i+1}, v)
5:     if suma < costo then
6:         costo ← suma
7:         vertice ← v
8:     end if
9: end for
10: return vertice
```

Para ver su implementación revisar anexo A.

Notar que el Algoritmo 3 Vértice Minimizador recibe como parte de la entrada la matriz de distancia de un grafo G, con la finalidad de no interferir con los tiempos de ejecución calculados para cada algoritmo propuesto en este estudio. Sea $|V| = n$. En cada iteración $i = 1, \dots, n-1$, el algoritmo calcula

$$s_i = \arg\min_{v \in V} \{A[s_{i-1}, v] + A[r_i, v] + A[r_{i+1}, v]\}.$$

Evaluar la expresión para un vértice v cuesta $\mathcal{O}(1)$ (tres accesos a A y sumas), y recorrer los n vértices cuesta $\mathcal{O}(n)$.

A continuación, se presenta el segundo y último algoritmo de este estudio, correspondiente al Algoritmo 4, el cual obtiene cada posición de la fuente a partir del Algoritmo 3, entregando el costo total acumulado y entregando la secuencia óptima de posiciones de fuentes.

Algoritmo 4 Algoritmo Windex 2

Entrada: Matriz de distancias A, índice de fuente inicial s_0, secuencia de requerimientos $req\_sequence$

Salida: Secuencia de fuentes y costo total

$secuencia \leftarrow [\,]$
$errores \leftarrow [\,]$
$s_{prev} \leftarrow s_0$
$costo\_total \leftarrow 0$
for $i \leftarrow 0$ **to** $|req\_sequence| - 2$ **do**
    $s_i \leftarrow$ Vértice minimizador$(A, s_{prev}, r_i, r_{i+1})$
    Agregar s_i a $secuencia$
    $costo\_total \leftarrow costo\_total + A[s_{prev}][s_i] + A[s_i][r_i]$
    $s_{prev} \leftarrow s_i$
end for
$r_{last} \leftarrow req\_sequence[|req\_sequence| - 1]$
$v^\star \leftarrow \arg\min_{u \in V} \left(A[s_{prev}][u] + A[u][r_{last}]\right)$
Agregar $v^\star$ a $secuencia$
$costo\_total \leftarrow costo\_total + A[s_{prev}][v^\star] + A[v^\star][r_{last}]$
return $(secuencia,\ costo\_total)$

Para ver su implementación revisar anexo A

Notar que el Algoritmo 4 Windex 2 recibe como parte de la entrada la matriz de distancia de un grafo G, con la finalidad de no interferir con los tiempos de ejecución calculados para cada algoritmo propuesto en este estudio.

Dado esto, se procede a demostrar la correctitud del Algoritmo 4, como consecuencia directa del Lema 19.

Proposición 20. *En un grafo mediano, el Algoritmo Windex 2 entrega la secuencia de fuentes que minimiza el costo del PLD.*

Demostración. Por el Lema 19, en una secuencia óptima cada s_i es Steiner Point de $\{s_{i-1}, r_i, s_{i+1}\}$, y éste es único en grafos medianos dada su definición mediante la propiedad USTP. El Algoritmo 4 de Windex 2, en cada paso $i < n$, calcula exactamente este Steiner Point usando (s_{i-1}, r_i, r_{i+1}), y finaliza con el vértice que minimiza $d(s_{n-1}, u) + d(u, r_n)$. □

A continuación, se presenta el análisis del tiempo de ejecución del Algoritmo 4.

Proposición 21 (Tiempo de ejecución de Windex 2)**.** *Sea G un grafo con n vértices y una secuencia de k requerimientos. El Algoritmo Windex 2 se ejecuta en tiempo $\mathcal{O}(kn)$.*

Demostración. En cada iteración $i = 1, \ldots, k-1$, el algoritmo realiza una llamada a la subrutina Vértice Minimizador que se ejecuta en tiempo $\mathcal{O}(n)$. El cierre final hacia r_k es otro $\arg\min$ sobre n vértices, también $\mathcal{O}(n)$. Por lo tanto, el tiempo total es $(k-1) \cdot \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(kn)$. □

Cabe señalar que en [HIK99] los autores muestran que los grafos medianos se pueden reconocer en tiempo subcuadrático $\mathcal{O}(n^{3/2} \log(n))$.

Capítulo 5

Implementaciones

En este capítulo se abordarán las implementaciones de los algoritmos estudiados en este trabajo de tesis, en donde se realizarán dos experimentos que serán detallados en la siguiente sección. Con estas pruebas se busca medir y contrastar el costo total obtenido por cada algoritmo, estimar empíricamente su radio de aproximación respecto al óptimo y evaluar cómo varía su comportamiento en función de la estructura del grafo, en particular de su densidad de aristas. De esta manera se puede confirmar empíricamente la Conjetura 1 en el caso de $k = 2$, sin descartar la aparición de posibles contraejemplos.

5.1. Diseño Experimental

En esta sección se describe la lógica general que se utilizó en los experimentos para evaluar el resultado de los algoritmos presentados en este trabajo. Lo que se quiere realizar es evaluar el resultado de tres algoritmos para el PLD: el Algoritmo 1 PLD Off-line, el cual entrega la solución óptima y se utiliza como referencia para comparar resultados con los algoritmos restantes; el Algoritmo 2 Glotón PLD (de Windex 1), que trabaja con una ventana de tamaño uno; y el Algoritmo 4 Windex 2, que trabaja con ventana de tamaño dos y aplica la estrategia basada en el Steiner Point.

Para las instancias de prueba se generaron secuencias aleatorias las cuales simulan secuencias de requerimientos de tamaño k, desde 10 hasta 100 aumentando de 10 en 10. Los tamaños fueron elegidos para que el cálculo del óptimo mediante el algoritmo Off-line fuera computacionalmente factible, pero lo suficientemente amplios como para extraer conclusiones relevantes. Para cada secuencia se utilizó una semilla fija que asegura la reproducibilidad, con la diferencia que las semillas no fueron las mismas entre los dos experimentos. Cada experimento se ejecutó cinco veces, cambiando la semilla en cada repetición para introducir variabilidad.

En cada experimento se registraron dos métricas principales: el costo total de la función objetivo para cada algoritmo y el radio de aproximación de los algoritmos aproximados respecto al óptimo. Todas las distancias necesarias fueron calculadas previamente y entregadas a los algoritmos en forma de matriz de distancias, evitando que el tiempo de cálculo de distancias influya

en la comparación. Cabe destacar que todas las métricas son particulares para cada grafo en estudio.

En el primer experimento se trabajó con grafos de diferentes parámetros para el Windex; en particular, se hace el experimento para árboles ($WX(T) = 2$), grafos completos de 6 vértices ($WX(K_6) = 6$), grafo completo de 4 vértices menos una arista, ciclo de 5 vértices y bipartito completo de 2 y 3 vértices. Estos últimos tres grafos corresponden a grafos que $WX(G) = \infty$. Se busca con este experimento concluír la influencia del parámetro del Windex según algoritmo o grafo, aumentando o disminuyendo su radio de aproximación.

En el segundo experimento se trabajó con grafos conexos generados por probabilidad de existencia por arista, con probabilidades $p = 0{,}3, 0{,}6$ y $0{,}9$. Esto se hace con el objetivo de generar grafos de diferente densidad y así concluir la influencia de la densidad de aristas del grafo con el comportamiento del radio de aproximación.

Ambos experimentos comparten el mismo planteamiento, diferenciándose únicamente en las características de los grafos empleados y de las semillas elegidas para las secuencias generadas, permitiendo analizar si la estructura del grafo favorece o perjudica de manera significativa a alguno de los algoritmos.

5.2. Experimento 1: parámetros del windex

Para este experimento se utilizaron las semillas $17, 25, 52, 89$ y 136, las cuales fueron elegidas sin un criterio en particular. Los grafos a evaluar no son significativamente grandes, dado que no se está estudiando densidad sino parámetros de windex, y estos grafos son: árbol binario balanceado de cuatro niveles, completo de 6 vértices, completo de 4 vértices sin una arista, ciclo de 5 vértices y completo bipartito dos-tres.

En este análisis se busca concluir acerca de la estructura de cada grafo y cómo se comportan para cada algoritmo, dado que tienen Windex $2, 6$ e ∞. Para esto, se mostrará a detalle los resultados para la primera semilla en gráficos y una tabla, para luego resumir los resultados de todas la semillas en una sola tabla. Cabe destacar que todos los gráficos contienen una la línea roja que representa $3/2$OPT, esto para poder establecer un límite en las conclusiones de la Conjetura 1 para el caso $k = 2$.

Se muestra en la Figura 5.1 el desempeño de cada algoritmo propuesto para el grafo de árbol binario balanceado de 4 niveles. Se puede observar que el Algoritmo 1 Off-Line coincide con el Algoritmo 4, lo cual se condice con la teoría demostrada en el Capítulo 4. Por otro lado, el Algoritmo 2 lo hace significativamente peor que el radio de aproximación estudiado de $3/2$, lo cual está dentro de lo esperado ya que el radio de aproximación del Algoritmo 2 es dos.

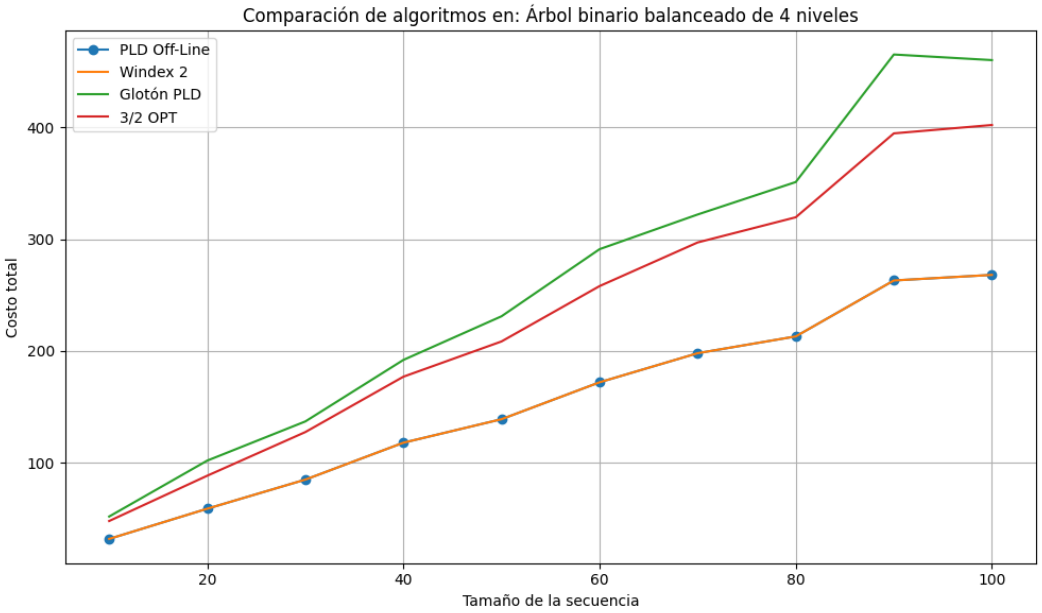


Figura 5.1

Se muestra en la Figura 5.2 el desempeño de cada algoritmo propuesto para el grafo completo de 6 vértices. Se puede observar que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo, sin embargo el desempeño de todos ellos se encuentran por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2.

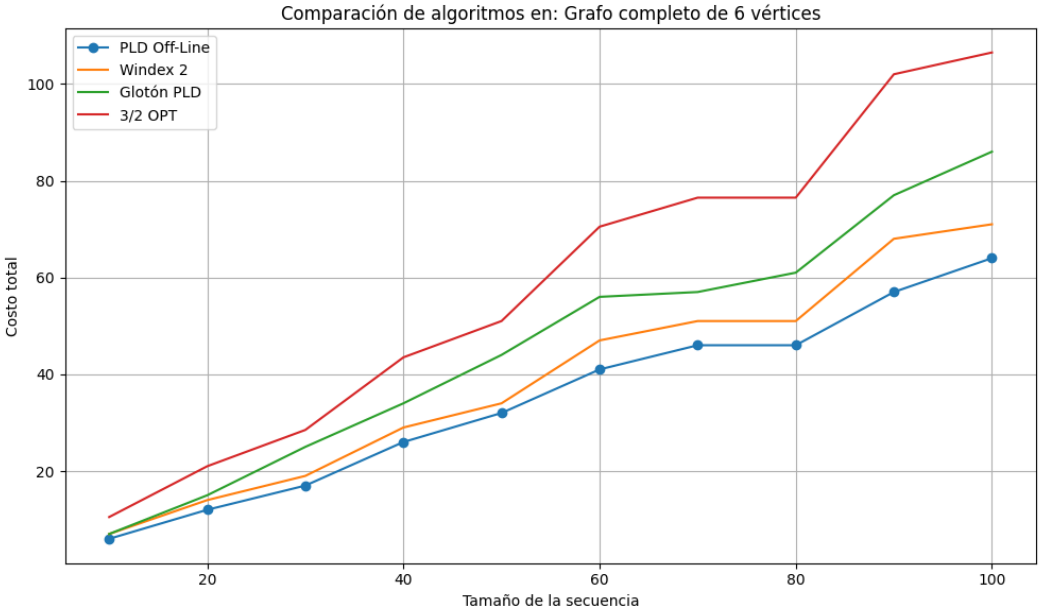


Figura 5.2

Se muestra en la Figura 5.3 el desempeño de cada algoritmo propuesto para el grafo completo de 4 vértices sin una arista. Se puede observar que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo se obtienen valores muy cercanos. Además, el desempeño de todos los algoritmos se encuentran por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2, inlcuso compartiendo valores óptimos.

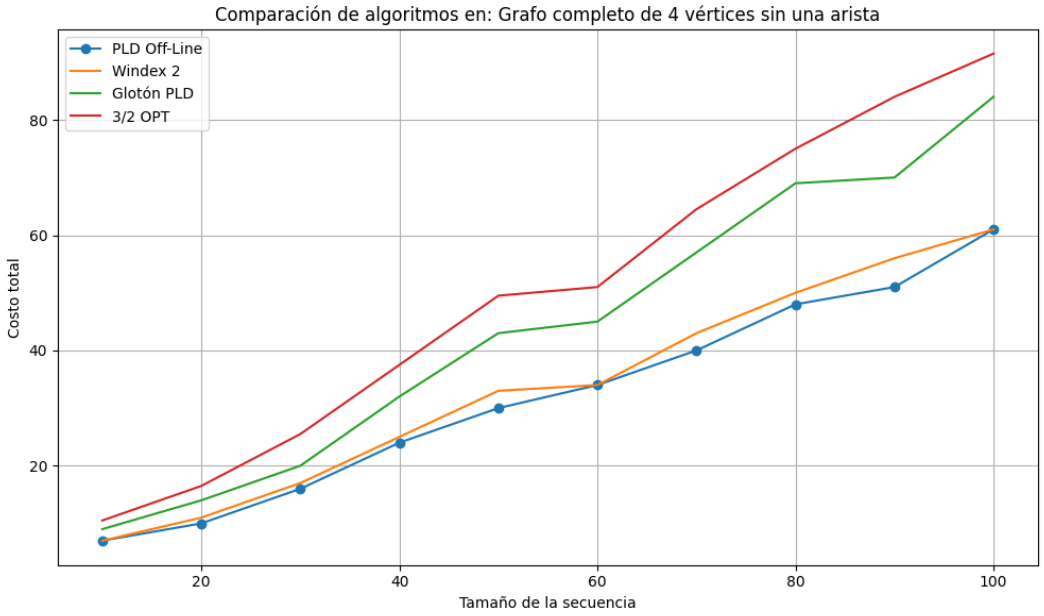


Figura 5.3

Se muestra en la Figura 5.4 el desempeño de cada algoritmo propuesto para el ciclo de 5 vértices. Se puede observar que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo se obtienen valores muy cercanos. Además, el desempeño de todos los algoritmos se encuentran en su mayoría por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2.

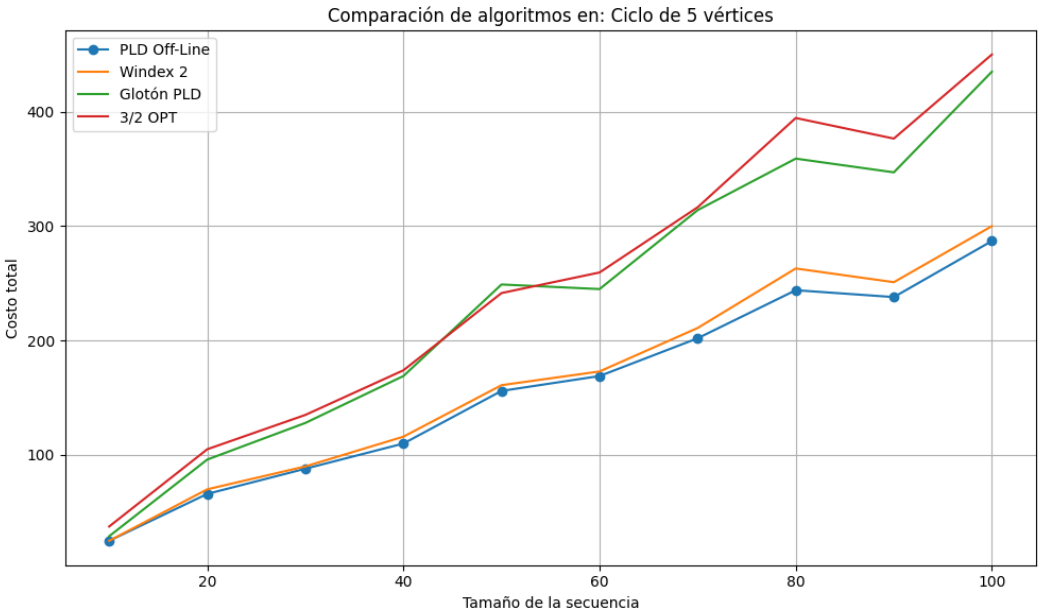


Figura 5.4

Se muestra en la Figura 5.5 el desempeño de cada algoritmo propuesto para el grafo bipartito completo dos-tres. Se puede observar que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo se obtienen valores muy cercanos, inlcuso compartiendo valores óptimos. Además, el desempeño de todos los algoritmos se encuentran en su mayoría por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2.

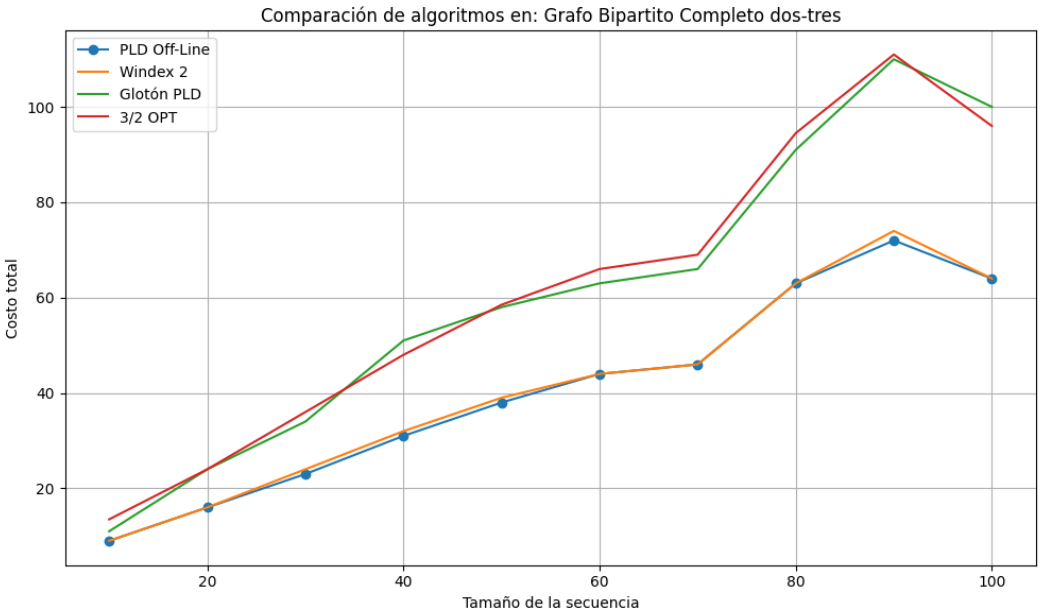


Figura 5.5

En el siguiente Cuadro 5.1 se presentan todos los valores obtenidos que fueron representados en los gráficos anteriores. Esto con la finalidad de calcular el radio de aproximación para el Algoritmo 4 de Windex 2 y para el Algoritmo 2.

Grafo	Tamaño	PLD Ópt.	Online		Glotón	
			Costo	Radio	Costo	Radio
$T_{2,4}$	10	32	32	1.000	52	1.625
	20	59	59	1.000	102	1.729
	30	85	85	1.000	137	1.612
	40	118	118	1.000	192	1.627
	50	139	139	1.000	231	1.662
	60	172	172	1.000	291	1.692
	70	198	198	1.000	322	1.626
	80	213	213	1.000	351	1.648
	90	263	263	1.000	465	1.768
	100	268	268	1.000	460	1.716
K_6	10	6	7	1.167	7	1.167
	20	12	14	1.167	15	1.250
	30	17	19	1.118	25	1.471
	40	26	29	1.115	34	1.308
	50	32	34	1.063	44	1.375
	60	41	47	1.146	56	1.366
	70	46	51	1.109	57	1.239
	80	46	51	1.109	61	1.326
	90	57	68	1.193	77	1.351
	100	64	71	1.109	86	1.344
$K_4 - e$	10	7	7	1.000	9	1.286
	20	10	11	1.100	14	1.400
	30	16	17	1.063	20	1.250
	40	24	25	1.042	32	1.333
	50	30	33	1.100	43	1.433
	60	34	34	1.000	45	1.324
	70	40	43	1.075	57	1.425
	80	48	50	1.042	69	1.438
	90	51	56	1.098	70	1.373
	100	61	61	1.000	84	1.377
C_5	10	25	25	1.000	29	1.160
	20	66	70	1.061	96	1.455
	30	88	90	1.023	128	1.455
	40	110	116	1.055	169	1.536
	50	156	161	1.032	249	1.596
	60	169	173	1.024	245	1.450
	70	202	211	1.045	314	1.554
	80	244	263	1.078	359	1.471
	90	238	251	1.055	347	1.458
	100	287	300	1.045	435	1.516
$K_{2,3}$	10	9	9	1.000	11	1.222
	20	16	16	1.000	24	1.500
	30	23	24	1.043	34	1.478
	40	31	32	1.032	51	1.645
	50	38	39	1.026	58	1.526
	60	44	44	1.000	63	1.432
	70	46	46	1.000	66	1.435
	80	63	63	1.000	91	1.444
	90	72	74	1.028	110	1.528
	100	64	64	1.000	100	1.563

Cuadro 5.1: Resultados experimentales para la semilla 17

Finalmente, se repite este mismo procedimiento para las semillas restantes. Se presenta el Cuadro 5.2 el cual rescata la secuencia con mayor radio de aproximación para cada grafo, cada algoritmo y cada semilla.

Semilla	Grafo	PLD Ópt.	Online (máx)		Glotón (máx)	
			Costo	Ratio	Costo	Ratio
17	$T_{2,4}$	268	268	1.000	465	1.768
	K_6	57	68	1.193	25	1.471
	$K_4 - e$	30	33	1.100	43	1.433
	C_5	244	263	1.078	249	1.596
	$K_{2,3}$	38	39	1.026	51	1.645
25	$T_{2,4}$	260	260	1.000	461	1.773
	K_6	12	15	1.250	7	1.400
	$K_4 - e$	39	39	1.000	55	1.528
	C_5	253	263	1.040	321	1.566
	$K_{2,3}$	51	52	1.020	104	1.576
52	$T_{2,4}$	155	155	1.000	279	1.800
	K_6	4	5	1.250	27	1.421
	$K_4 - e$	6	7	1.167	48	1.500
	C_5	24	29	1.208	136	1.511
	$K_{2,3}$	30	31	1.033	50	1.667
89	$T_{2,4}$	171	171	1.000	309	1.807
	K_6	5	6	1.200	8	1.600
	$K_4 - e$	21	23	1.095	37	1.762
	C_5	149	154	1.034	227	1.523
	$K_{2,3}$	60	61	1.017	95	1.583
136	$T_{2,4}$	293	293	1.000	510	1.741
	K_6	25	30	1.200	8	1.600
	$K_4 - e$	16	18	1.125	18	1.500
	C_5	49	49	1.000	74	1.510
	$K_{2,3}$	64	67	1.047	108	1.688

Cuadro 5.2: Radio mayor por algoritmo y grafo para todas las semillas. Vemos que para el caso Algoritmo Windex 2, el mayor radio de aproximación registrado fue de $1{,}250$ en K_6 y para el caso del Algoritmo Glotón es $1{,}807$ en $T_{2,3}$.

En conclusión del experimento, se puede observar dado los valores del Cuadro 5.2 que para todas las semillas, el radio de aproximación del Algoritmo 2 Glotón es mayor a $3/2$ en todas las ocasiones sólo para los árboles. Por otro lado, en todas las semillas ocurre que mientras mayor es el valor del Windex, más cerca del óptimo se encuentra, a excepción de los árboles que da con exactitud. El grafo completo, por el contrario, es el que obtuvo mayor radio de aproximación para todas las semillas en referencia al Algoritmo 4 Windex 2.

5.3. Experimento 2: Densidad del Grafo

Para este experimento se utilizaron las semillas $7, 28, 46, 139$ y 258, las cuales fueron elegidas sin un criterio en particular. Los grafos a evaluar son notablemente más grandes que los evaluados en el experimento anterior, dado que ahora se está estudiando el comportamiento de los algoritmos según la densidad, y estos son grafos de 100 nodos con probabilidad de arista $0{,}3, 0{,}6, 0{,}9$.

En este análisis se busca concluir acerca de la densidad de cada grafo y cómo se comportan para cada algoritmo. Para esto, se mostrará a detalle los resultados para la primera semilla en gráficos y una tabla, para luego resumir los resultados de todas la semillas en una sola tabla. Cabe destacar que todos los gráficos contienen una la línea roja que representa $3/2$OPT, esto para poder establecer un límite en las conclusiones de la Conjetura 1 para el caso $k = 2$.

Se muestra en la Figura 5.6 el desempeño de cada algoritmo propuesto para el grafos de 100 nodos con probabilidad de arista $0{,}3$. Se puede observar que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo el desempeño de todos ellos se encuentran por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2.

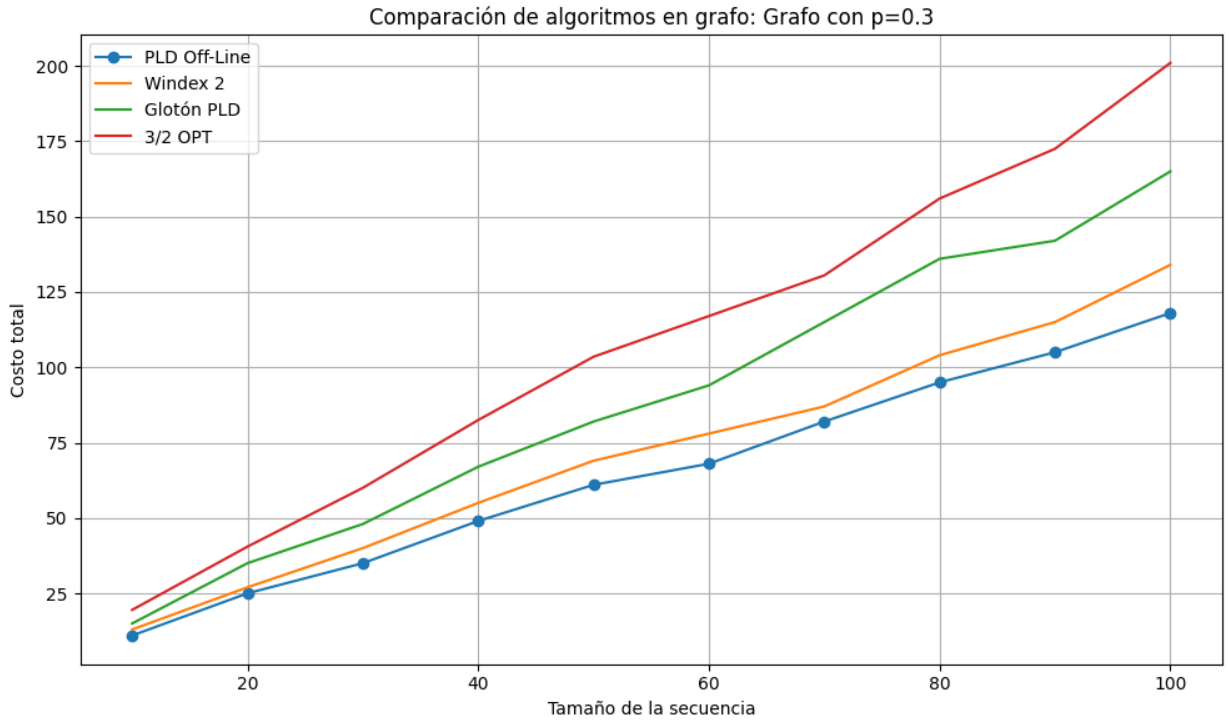


Figura 5.6

Se muestra en la Figura 5.7 el desempeño de cada algoritmo propuesto para el grafos de 100 nodos con probabilidad de arista $0{,}6$. Se puede observar nuevamente que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo el desempeño de

todos ellos se encuentran por debajo del radio de aproximación $3/2$, en donde el Algoritmo 4 está significativamente más cercano a los valores óptimos que el Algoritmo 2. Cabe destacar que el desempeño del Algoritmo 2 se nota un poco menor que en la Figura 5.6.

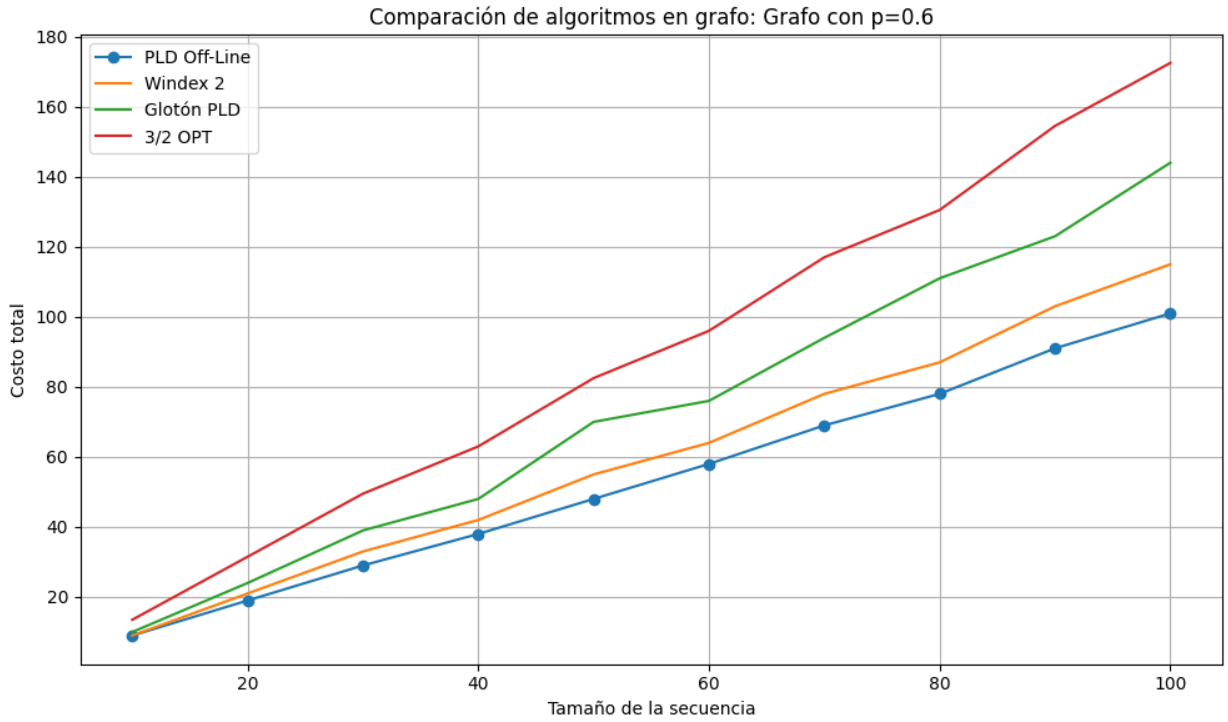


Figura 5.7

Se muestra en la Figura 5.8 el desempeño de cada algoritmo propuesto para el grafos de 100 nodos con probabilidad de arista $0{,}9$. Se puede observar nuevamente que el Algoritmo 1 Off-Line no coincide con ningún otro algoritmo en su totalidad, sin embargo el desempeño de todos ellos se encuentran por debajo del radio de aproximación $3/2$, con la diferencia que el Algoritmo 2 Glotón y Algoritmo 4 está significativamente más cercanos a los valores óptimos, con un aumento significativo en el desempeño del Algoritmo 2 Glotón.

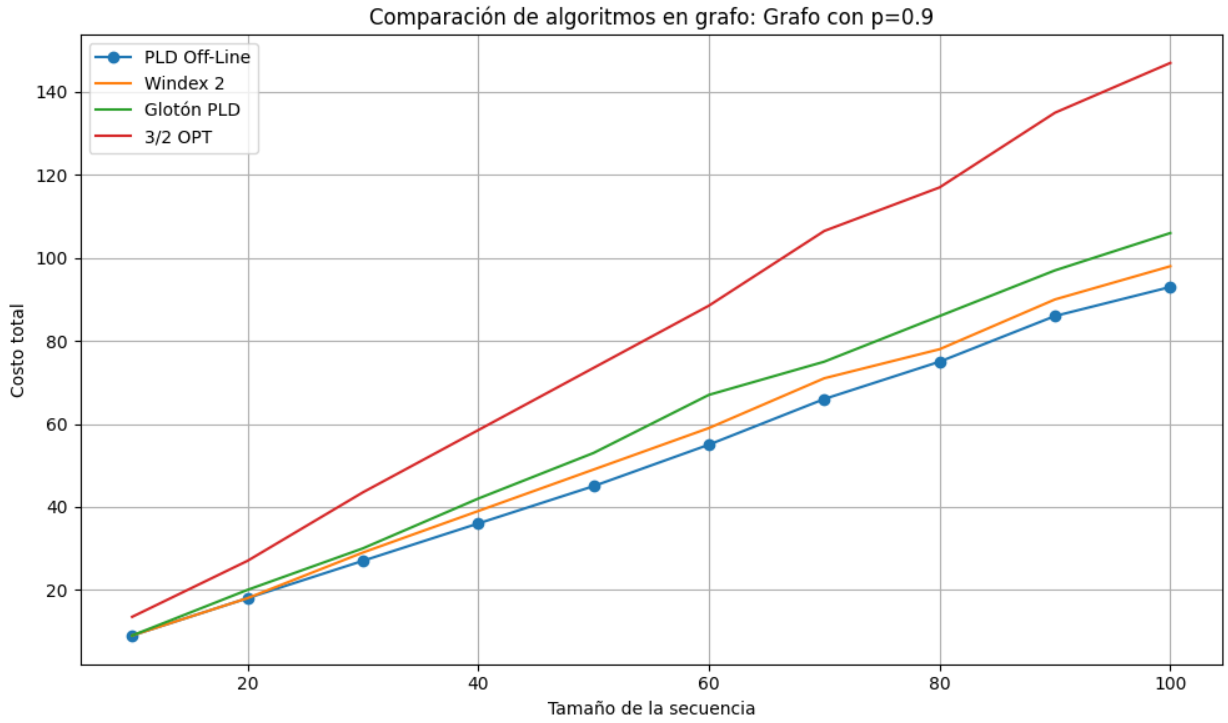


Figura 5.8

En el siguiente Cuadro 5.3 se presentan todos los valores obtenidos que fueron representados en los gráficos anteriores. Esto con la finalidad de calcular el radio de aproximación para el Algoritmo 4 de Windex 2 y para el Algoritmo 2.

Grafo	Tamaño	PLD Ópt.	Online		Glotón	
			Costo	Radio	Costo	Radio
p=0.3	10	11	12	1.091	16	1.455
	20	23	24	1.043	31	1.348
	30	36	40	1.111	48	1.333
	40	46	54	1.174	64	1.391
	50	57	63	1.105	76	1.333
	60	69	77	1.116	98	1.420
	70	84	92	1.095	116	1.381
	80	96	109	1.135	138	1.438
	90	108	127	1.176	156	1.444
	100	121	137	1.132	171	1.413
p=0.6	10	10	12	1.200	15	1.500
	20	20	21	1.050	27	1.350
	30	31	34	1.097	45	1.452
	40	39	43	1.103	53	1.359
	50	48	56	1.167	62	1.292
	60	58	67	1.155	82	1.414
	70	67	76	1.134	94	1.403
	80	79	91	1.152	108	1.367
	90	89	95	1.067	118	1.326
	100	103	108	1.049	139	1.350
p=0.9	10	9	9	1.000	11	1.222
	20	18	19	1.056	21	1.167
	30	28	30	1.071	33	1.179
	40	38	39	1.026	45	1.184
	50	47	49	1.043	53	1.128
	60	55	59	1.073	65	1.182
	70	63	67	1.063	70	1.111
	80	75	78	1.040	87	1.160
	90	84	88	1.048	98	1.167
	100	94	98	1.043	104	1.106

Cuadro 5.3: Resultados para los grafos con probabilidades $p = 0{,}3$, $p = 0{,}6$ y $p = 0{,}9$.

Finalmente, se repite este mismo procedimiento para las semillas restantes. Se presenta el Cuadro 5.4 el cual rescata la secuencia con mayor radio de aproximación para cada grafo, cada algoritmo y cada semilla.

Semilla	Grafo	PLD Ópt.	Online (máx)		Glotón (máx)	
			Costo	Ratio	Costo	Ratio
7	p=0.3	107	125	1.168	152	1.421
	p=0.6	93	102	1.097	128	1.376
	p=0.9	93	97	1.043	106	1.140
28	p=0.3	108	127	1.176	169	1.457
	p=0.6	72	78	1.097	105	1.458
	p=0.9	36	39	1.083	91	1.230
46	p=0.3	117	132	1.205	164	1.556
	p=0.6	40	43	1.188	59	1.475
	p=0.9	46	48	1.059	103	1.198
139	p=0.3	23	27	1.174	35	1.522
	p=0.6	48	55	1.146	71	1.479
	p=0.9	36	39	1.083	42	1.167
258	p=0.3	118	134	1.136	165	1.398
	p=0.6	101	115	1.139	144	1.426
	p=0.9	45	49	1.089	53	1.178

Cuadro 5.4: Radio mayor por algoritmo y grafo para todas las semillas. Vemos que para el caso Algoritmo Windex 2, el mayor radio de aproximación registrado fue de $1{,}205$ en el grafo con probabilidad de arista $p = 0{,}3$ y para el caso del Algoritmo Glotón es $1{,}556$ en este mismo grafo.

En conclusión del experimento, se puede observar dado los valores del Cuadro 5.4 que para todas las semillas, mientras más denso sea el grafo, el Algoritmo 2 Glotón se acerca más a los valores del óptimo. Este fenómeno es independiente de la cantidad de vértices del grafo, dado que se probó para las mismas probabilidades pero con 30 vértices, obteniendo curvas muy similares.

5.4. Conclusiones y Trabajo Futuro

Ambos experimentos muestran que Algoritmo 4 Windex 2 supera consistentemente al Algoritmo 2 Glotón en términos de radio de aproximación, especialmente en grafos con $WX(G) = 2$ o alta densidad. La principal diferencia observada es la mayor sensibilidad del Algoritmo 2 Glotón a la estructura del grafo, con aumento en los radios de forma significativa en grafos con $WX(G) = \infty$.

Los datos respaldan la Conjetura 1 para el caso $k = 2$ en un amplio conjunto de casos, sin que se hayan detectado contraejemplos en las instancias evaluadas. Por otro lado, en los experimentos realizados no se han encontrado contraejemplos, observándose radios de aproximación consistentemente menores o iguales a $1{,}5$ en las clases de grafos estudiadas. Este resultado motiva trabajos futuros orientados a extender el análisis a valores mayores de k y a explorar posibles mejoras en los radios obtenidos.

Por otro lado, dentro de las limitaciones de este experimento se encuentra el bajo control de las secuencias generadas, dado que no se hizo diferencia entre secuencias de mayor acceso o mayor dificultad, siendo esto un posible factor en el aumento de los radios de aproximación. Se propone entonces para investigaciones futuras que:

- Encontrar un procedimiento de crear secuencias convenientes con el objetivo de obtener el mayor radio de aproximación posible en cada grafo.
- Investigar y repetir los experimentos para algoritmos que trabajen dentro de ventanas de mayor tamaño, comenzando por $WX(G) = 3$.

Bibliografía

[CGS87] F. R. K. Chung, R. L. Graham, and M. E. Saks. Dynamic search in graphs. In *Discrete Algorithms and Complexity*, pages 352–387. Academic Press, 1987. 2, 9, 24, 27, 32, 38, 42

[CGS89] Fan Chung, Ronald Graham, and Michael Saks. A dynamic location problem for graphs. *Combinatorica*, 9:111–131, 1989. 2, 3, 6, 7, 9, 12, 13, 14, 16, 24, 27, 32, 34, 38

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 4th edition, 2022. 23

[HIK99] Johann Hagauer, Wilfried Imrich, and Sandi Klavžar. Recognizing median graphs in subquadratic time. *Theoretical Computer Science*, 215(1-2):123–136, 1999. 44

[LLRKS85] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1985. 6

[OM20] Marek Okulewicz and Jacek Mańdziuk. Monte carlo tree search for the dynamic vehicle routing problem with time windows. *Expert Systems with Applications*, 159:113600, 2020. 6

[Psa13] Harilaos N. Psaraftis. Dynamic vehicle routing problems. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, pages 223–248. Society for Industrial and Applied Mathematics, 2013. 6

[Wil10] Daniel J. Wildstrom. Resource relocation on asymmetric networks. *Journal of Graph Algorithms and Applications*, 14(2):149–163, 2010. 7

[ZPBJ22] Yunlong Zhang, Marco Pavone, Dimitris Bertsimas, and Patrick Jaillet. Stochastic and dynamic vehicle routing with uncertain demand and travel times. *Operations Research*, 70(1):184–201, 2022. 6

Apéndice A

Código

Código referente al Algoritmo 1 Off-Line óptimo.

```
def algoritmo_pld_optimo(A, secuencia):
    s0_index = secuencia[0]
    r_indices = secuencia[1:]
    n = A.shape[0]
    k = len(r_indices)

    memoria = np.zeros((n, k, 2), dtype=object)

    rk = r_indices[-1]
    for i in range(n):
        memoria[i][k-1][0] = A[i][rk]
        memoria[i][k-1][1] = i

    for j in reversed(range(1, k - 1)):
        rj = r_indices[j]
        for i in range(n):
            l_opt, costo_opt = min(
                ((l, A[i][l] + A[rj][l] + memoria[l][j + 1][0]) for l in range(n)),
                key=lambda x: x[1]
            )
            memoria[i][j][0] = costo_opt
            memoria[i][j][1] = l_opt

    r1 = r_indices[0]
    for i in range(n):
        memoria[i][0][0] = A[s0_index][i] + A[r1][i] + memoria[i][1][0]

    s1_index = np.argmin([memoria[i][0][0] for i in range(n)])
    for i in range(n):
        memoria[i][0][1] = s1_index
```

```
    secuencia_optima = [s1_index]
    v = s1_index
    for j in range(1, k):
        v = memoria[v][j][1]
        secuencia_optima.append(v)

    costo_total = float(memoria[s1_index][0][0])
    return secuencia_optima, costo_total
```

Código referente al Algoritmo 2 Glotón (de windex 1).

```
def algoritmo_gloton(G, seq):
    r_indices = seq[1:]
    secuencia = []
    costo_total = 0
    prev = seq[0]
    for ri in r_indices:
        si = ri
        secuencia.append(int(si))
        costo_total += G[prev][si]
        prev = si
    return secuencia, float(costo_total)
```

Código referente al Algoritmo 3 vértice minimizador.

```
def algoritmo_vertice_minimizador(A, s_prev, r_i, r_next):
    n = A.shape[0]
    costo = float('inf')
    vertice = None
    unico = True

    for v in range(n):
        suma = A[s_prev][v] + A[r_i][v] + A[r_next][v]
        if suma < costo:
            costo = suma
            vertice = v
            unico = True
        elif suma == costo:
            unico = False

    return vertice
```

Código referente al Algoritmo 4 de Windex 2.

```
def algoritmo_online_completo(A, req_sequence):
    secuencia = []
    errores = []
```

```
s_prev = int(req_sequence[0])
costo_total = 0.0

for i in range(1, len(req_sequence) - 1):
    r_i = int(req_sequence[i])
    r_next = int(req_sequence[i + 1])

    algoritmo_vertice_minimizador
    v = algoritmo_vertice_minimizador(A, s_prev, r_i, r_next)
    if v is None:
        errores.append((i, r_i, r_next))
        continue

    secuencia.append(int(v))
    costo_total += A[s_prev][v] + A[v][r_i]
    s_prev = v

r_last = int(req_sequence[-1])
v = int(np.argmin([A[s_prev][u] + A[r_last][u] for u in range(A.shape[0])]))
secuencia.append(v)
costo_total += A[s_prev][v] + A[v][r_last]

return secuencia, errores, float(costo_total)
```