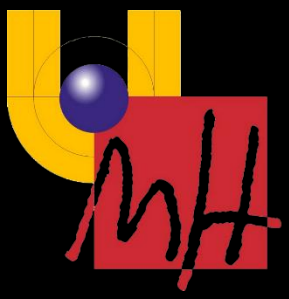
UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN



"OPTIMIZACIÓN DE RUTAS Y DISTRIBUIDORES EN REDES DE TELECOMUNICACIÓN"

TRABAJO FIN DE GRADO

Julio – 2025

AUTOR: Juan Francisco Martínez López

DIRECTOR: José Luis Sainz-Pardo Auñón

ÍNDICE

1. INTRODUCCIÓN 8

1.1. CONTEXTUALIZACIÓN Y MOTIVACIÓN 8

1.2. PROBLEMAS EN EL DISEÑO DE REDES 15

1.3. MÉTODOS PARA EL DISEÑO DE REDES 18

1.4. PROBLEMA DE LOCALIZACIÓN DE SWITCHES 21

1.5. ESTADO DEL ARTE 24

1.6. OBJETIVOS DEL PROYECTO 28

1.7. ESTRUCTURA DEL DOCUMENTO 29

2. MATERIAL Y MÉTODOS 31

2.1. MODELO MATEMÁTICO 31

2.2. APORTACIONES Y ALGORITMOS HEURÍSTICOS 33

2.3. PRIM CLÁSICO CON COSTE DE NODOS 34

2.4. PRIM DINÁMICO 36

2.5. HEURÍSTICO N1 38

2.6. HEURÍSTICO N2 40

2.7. HEURÍSTICO N3 43

2.8. MÉTODO GENÉTICO 45

3. RESULTADOS Y DISCUSIÓN 48

3.1. DATOS EXPERIMENTALES 48

3.2. HERRAMIENTAS DE PROGRAMACIÓN Y ANÁLISIS 51

3.3. PRESENTACIÓN DE LOS RESULTADOS Y ANÁLISIS COMPARATIVO 54

4. CONCLUSIONES 64

4.1. CONCLUSIONES GENERALES 64

4.2. TRABAJOS FUTUROS Y PROPUESTAS DE MEJORA 66

5. ANEXOS 67

5.1. CÓDIGOS DE LOS ALGORITMOS DESARROLLADOS 67

5.2. TABLAS DE RESULTADOS 86

6. BIBLIOGRAFÍA 107

RESUMEN

En la actualidad, la rapidez de acceso a diversos sitios web y la ubicación óptima de los servidores son prioridades fundamentales para cualquier proveedor de servicios de Internet (ISP). En este contexto, resulta crucial analizar la cantidad de equipos o nodos por los que deben transitar los datos, tanto en la ruta de ida como en la de vuelta, debido a que cada equipo intermedio que forme parte puede incrementar el retardo en términos de latencia y los costes operativos de la red en relación con la cantidad de saltos hasta llegar al destino.

Este proyecto se centra en el desarrollo e innovación de varios algoritmos de optimización de rutas y switches, con el objetivo principal de minimizar el coste de ambos. Se considera que los interruptores, conmutadores o switches son aquellos nodos con más de dos conexiones, y su reducción contribuye a abaratar la topología de la red en términos operativos, favoreciendo el uso de dispositivos con una entrada y una única salida que actúan únicamente como puntos de tránsito masivo de datos.

Para validar estos algoritmos, se han realizado simulaciones utilizando instancias de grafos estructurados. Los resultados obtenidos permiten evaluar de manera cuantitativa el impacto de cada variante del algoritmo en términos de coste total. Se ha observado que algunas adaptaciones heurísticas y el enfoque genético proporcionan mejoras significativas frente a la versión clásica del algoritmo de Prim añadiendo la variable del coste por disponer de un switch en el árbol.

Estos estudios revelan que, en entornos con alta complejidad y grandes volúmenes de datos, la estrategia genético-heurística resulta particularmente efectiva. En este sentido, el proyecto abre nuevas líneas de investigación para la integración de estas técnicas en entornos reales de redes de telecomunicaciones y para la aplicación de métodos avanzados de paralelización y optimización algorítmica.

ABSTRACT

Currently, the speed of access to various websites and the optimal placement of servers are key priorities for any Internet Service Provider (ISP). In this context, it is crucial to analyse the number of hosts or nodes through which the data must pass, both on the outgoing and on the return route, because each intermediate host that is part of it can increase the delay in terms of latency and the operational costs of the network in relation to the number of hops to reach the destination.

This project focuses on the development of several route- and switch-optimization algorithms, with the primary goal of minimizing the number of switches required in the network. Switches are defined as those nodes with more than two connections, and reducing their number helps lower the network's operational topology costs by favoring the use of devices with a single input and a single output that serve solely as high-volume data transit points.

To validate these algorithms, simulations have been carried out using instances of structured networks. The results obtained allow a quantitative evaluation of the impact of each variant of the algorithm in terms of total cost. It has been observed that some heuristic adaptations and the genetic approach provide significant improvements over the classical version of Prim's algorithm by adding the variable of the cost of having a switch in the tree.

These studies reveal that, in environments with high complexity and large volumes of data, the genetic-heuristic strategy is particularly effective. In this sense, the project opens up new lines of research for the integration of these techniques in real telecommunications network environments and for the application of advanced parallelisation and algorithmic optimisation methods.

1. INTRODUCCIÓN

1.1. CONTEXTUALIZACIÓN Y MOTIVACIÓN

En la última década, la integración de Internet en la sociedad ha alcanzado niveles sin precedentes. Como se puede apreciar en la Figura 1, la media poblacional mundial de uso de Internet es de un 67.9%, convirtiéndose en la piedra angular de la economía global y de las relaciones sociales. El acceso a servicios en la nube, aplicaciones web, plataformas de streaming y redes sociales obliga a los Internet Service Provider (ISP) a garantizar no sólo conectividad, sino también niveles bajos de latencia, alta disponibilidad y una calidad de servicio uniforme para todo tipo de usuarios. En este contexto, la topología de red que subyace a la infraestructura de red de este tipo de empresas se convierte en un activo estratégico, pues define la ruta que recorrerán los paquetes de datos desde el usuario que realiza la petición de un determinado servicio hasta los servidores que alojan el contenido, y viceversa. Cada nodo intermedio, y en particular cada dispositivo switch de capa tres, capaz de reenrutar tráfico en función del tránsito de la red, introduce un retardo adicional en la transmisión, incrementando el coste de la infraestructura y los requisitos de mantenimiento.

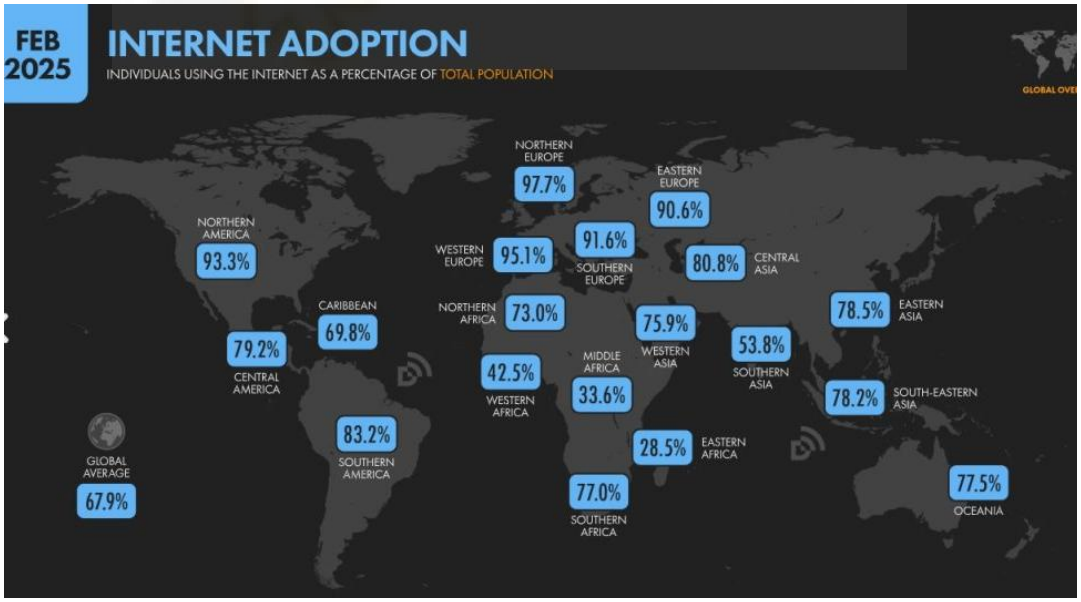


Figura 1. Regiones con mayor porcentaje de uso de Internet.

Extraída de: https://marketing4ecommerce.net/usuarios-de-internet-mundo/

Para comprender la relevancia de este problema, se puede observar en la Figura 2 el crecimiento tan grande de la demanda de ancho de banda; según el "Cisco Visual Networking Index" [1], el tráfico de datos global creció un 26% anual entre 2017 y 2022, y se proyecta que, para 2025, el tráfico global supere los 4,8 zettabytes. Este fenómeno obedece a la proliferación de servicios de video en muy alta definición, reuniones a través de videoconferencias, aplicaciones de Internet de las Cosas (IoT) y la virtualización de centros de datos. Los ISP, que en origen se limitaban a unir nodos periféricos con troncales de alta capacidad, han evolucionado hacia arquitecturas más densas y complejas, con múltiples puntos de intercambio (IXPs), servidores caché distribuidos y enlaces redundantes para garantizar la continuidad del servicio.

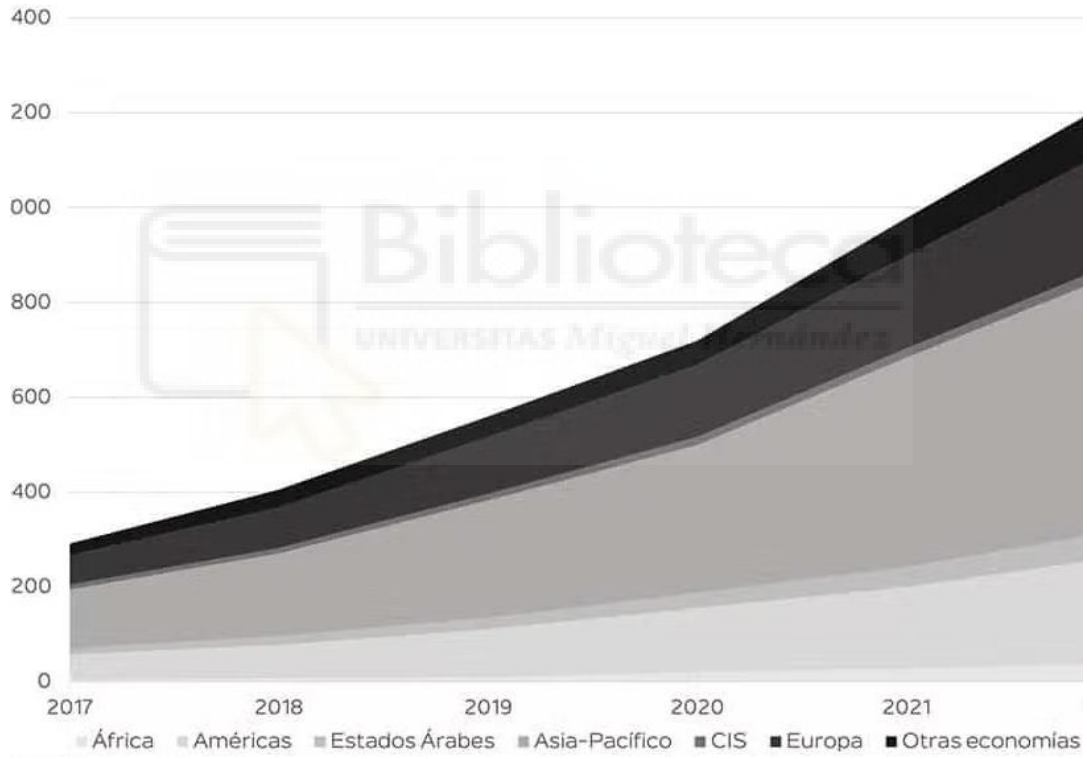


Figura 2. Consumo de ancho de banda internacional por región.

Extraído de: https://www.stackscale.com/es/blog/internet-evolucion-estadisticas/

Históricamente, los algoritmos de optimización de redes se han centrado en minimizar el coste de los enlaces, sin incluir en su función objetivo un aumento de complejidad y coste operativo que implica la instalación de nodos con múltiples entradas y salidas. El problema del árbol de expansión mínima (MST) es un buen ejemplo. A partir de un grafo ponderado en el que cada arista representa un enlace entre dos nodos con un coste asociado, el objetivo es conectar todos los nodos utilizando un subconjunto de

aristas de forma que la suma de los pesos sea mínima tal y como se muestra en la Figura 3. Algoritmos como los de Prim [2] y Kruskal [3] aportan soluciones eficientes para este problema. Sin embargo, al buscar únicamente la minimización del peso de las aristas, se desatiende el hecho de que cada nodo puede requerir asumir un switch si se precisa de más de dos enlaces en ese punto. En un entorno ISP real, puede suponer un punto crítico en caso de una avería. Es, por tanto, importante, centrar esfuerzos en analizar cuántos dispositivos de este tipo son necesarios, y en calcular cuánta redundancia merece según el porcentaje de importancia para la red.

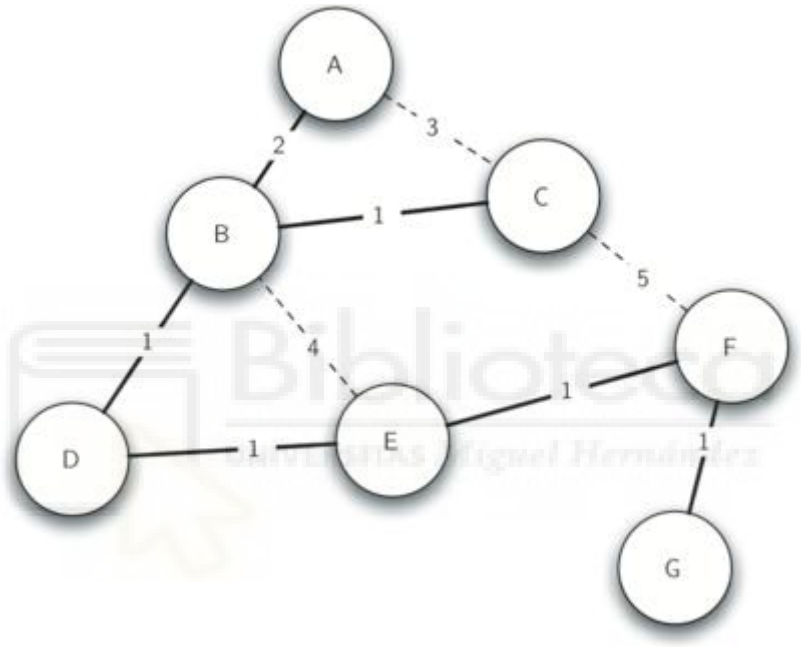


Figura 3. Árbol de expansión mínimo de un grafo.

Extraído de: https://runestone.academy/ns/books/published/pythoned/Graphs/AlgoritmoDePrimDelArbolDeExpansion.html

Para mitigar esta carencia, en los últimos años han surgido líneas de investigación que incorporan explícitamente el coste de los nodos al problema de optimización. En este modelo extendido, cada nodo recibe un peso que refleja el coste asociado a instalar y mantener un switch que atienda todas las conexiones de ese nodo. Dicha aproximación se enmarca en el problema conocido como “Minimum Steiner Tree with Node Weights” [4] o variaciones del “Degree-Constrained Minimum Spanning Tree” [5]. En la práctica, esto implica redefinir la función objetivo. No es suficiente con minimizar la suma de costes de las aristas, sino que hay que minimizar la suma de costes de enlaces junto con la suma de pesos de los nodos que actúan como switches. De esta

manera, se promueve el uso de dispositivos de capa dos, que son infraestructuras dedicadas al transporte masivo de información mientras, por otro lado, se penaliza la concentración excesiva de enlaces en un único switch de alto desempeño.

En un ISP, esta penalización tiene implicaciones muy concretas. En primer lugar, reduce el número total de switches de agregación o núcleo, resaltando que cada switch de capa tres exige un nivel de redundancia, mantenimiento e inversión superior al de un dispositivo menos complejo. Además, la reducción del número de switches disminuye el coste energético. Teniendo en cuenta que los centros de procesamiento de datos (CPD) en 2020 consumieron alrededor del 1% del suministro eléctrico mundial [6], se estima que los dispositivos de red representan aproximadamente el 15% de ese consumo.

Desde el punto de vista operativo, se simplifica notablemente cuando la red tiene menos puntos críticos de falla. Cada switch representa un punto potencial de interrupción: si un switch de alta densidad deja de funcionar, puede desencadenar en problemas de balanceo de carga, cortes en el servicio por conmutaciones a través de caminos secundarios, y desconexiones totales a usuarios y servidores, según su ubicación.

Otro punto a tener en cuenta proviene de la ubicación geográfica de los centros de datos y puntos de intercambio. Los ISP líderes, tanto a nivel global como pueden ser Lumen, GTT, Telia o Verizon, como a nivel nacional como son Movistar, Vodafone, Orange o Aire Networks, han desplegado multitud de Data Centers y enlaces para acercar contenido al usuario final [7] [8], como se aprecia en la Figura 4 con el despliegue de fibra realizada por Lumen. No obstante, la topología de interconexión entre dichos centros y las redes metropolitanas no siempre está optimizada. Por ejemplo, un ISP que conecte directamente un Data Center en Madrid con otros en Barcelona, Sevilla y Lisboa puede elegir rutas triangulares que minimicen el retardo de enlace, pero que obliguen a ciertos nodos intermedios a actuar como switches enrutables si no se analiza el coste de los nodos, ya que se tiende a generar redes malladas con la finalidad de garantizar la mayor calidad de servicio.

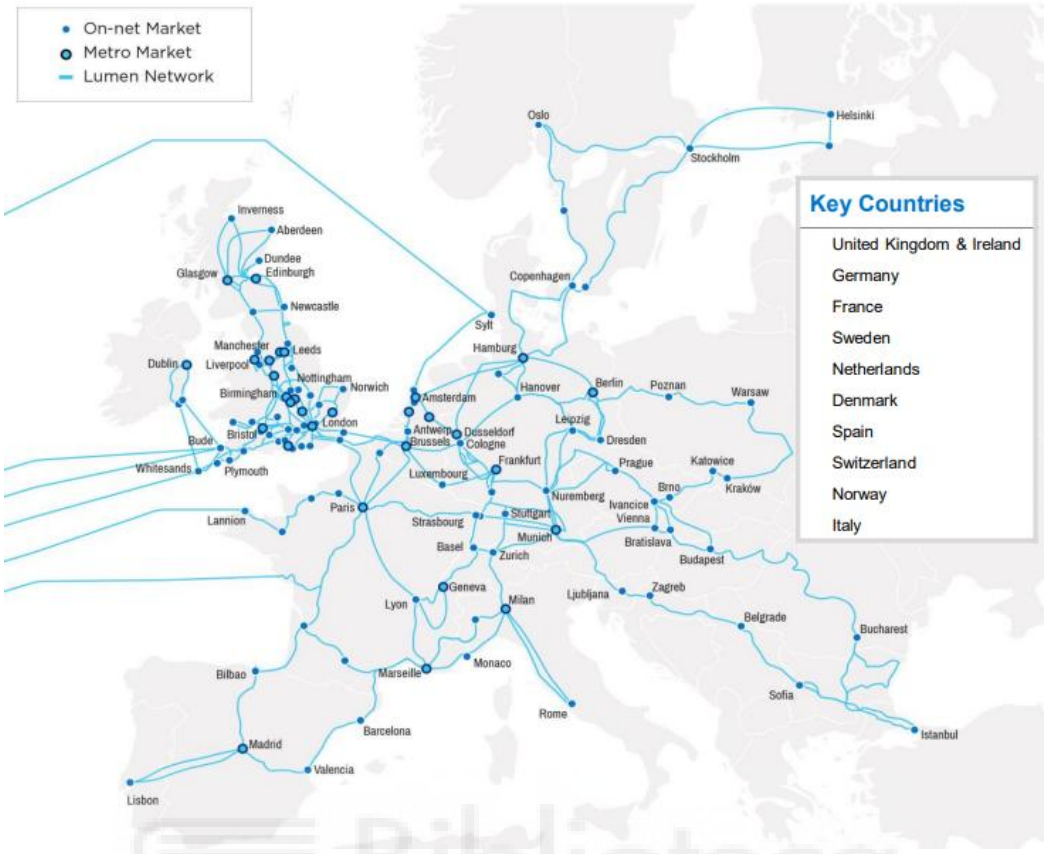


Figura 4. Lumen, despliegue de fibra en la red europea.

Extraído de: https://www.datacenterdynamics.com/es/noticias/lumen-vender%C3%A1-su-negocio-de-emea-a-colt-por-1800-millones-de-d%C3%B3lares/

Varios programas de investigación han abordado el estudio de diseño de redes backbone para ISP, analizando la ubicación óptima de routers principales y enlaces de alto tránsito bajo restricciones de presupuesto y requisitos de capacidad [9] o, de manera similar, la creación de técnicas de enrutamiento con garantía de QoS, considerando ancho de banda, latencia y jitter [10].

Por otro lado, la evolución de las redes definidas por software (SDN) y la virtualización de funciones de red (NFV) introducen nuevos retos y oportunidades en la optimización topológica como se muestran en las Figuras 5 y 6. En entornos SDN, el controlador centralizado dispone de una visión global de la red y puede reconfigurar rutas en tiempo real [13]. Sin embargo, si el algoritmo de cálculo de rutas no considera factores determinados por la carga de tránsito que está soportando un switch core concreto, el controlador puede elegir rutas correctas basadas en términos de salto, pero que pasen por equipos con alto nivel de saturación. La motivación de optimizar la topología para

distribuir la carga, cobra mayor relevancia en SDN, donde los cambios deben igualmente equilibrar múltiples objetivos (latencia, coste, resiliencia). NFV, por su parte, desplaza funciones de red como configuraciones de firewall, NAT, o balanceadores, hacia instancias virtualizadas que pueden residir en servidores genéricos.

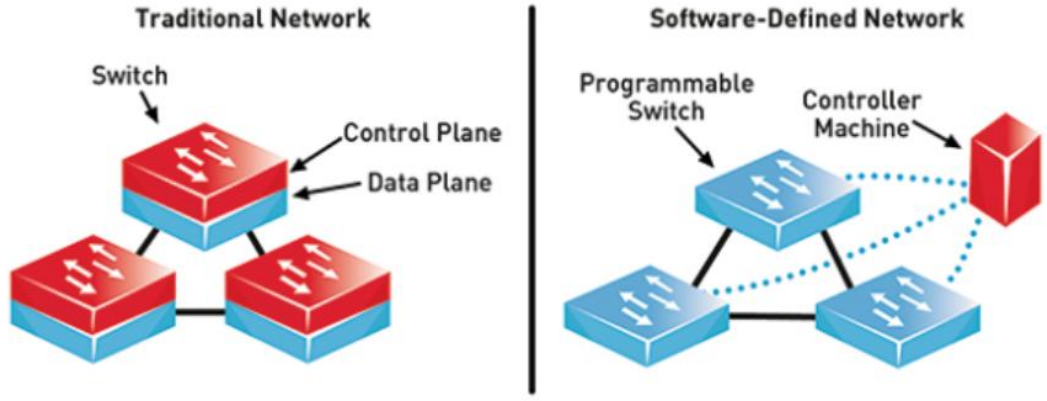


Figura 5. Diferenciación entre una red convencional y una red SDN.

Fuente: https://blogs.salleurl.edu/en/what-software-defined-networking-sdn

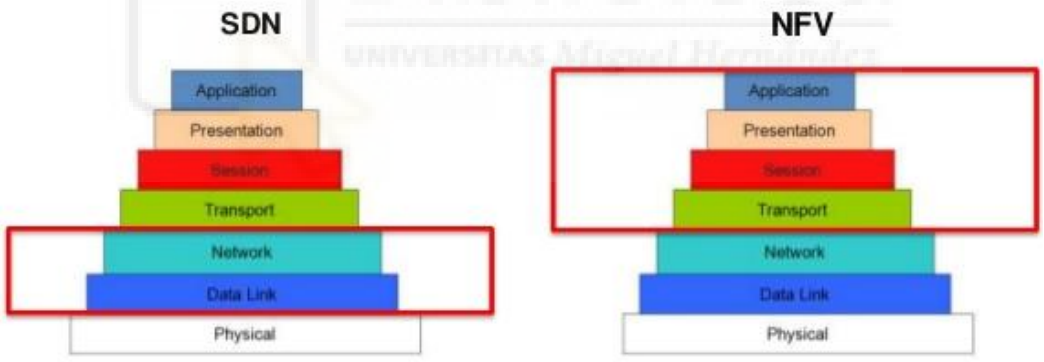


Figura 6. Diferenciación en la optimización de las capas del modelo OSI en una red SDN y en una red NFV.

Extraído de: https://comparacloud.com/servicios-clouds/sdn-y-nfv/

El problema resultante pertenece a la familia de problemas NP-Completeness [14] los cuales se centran en encontrar un árbol de expansión mínima que disminuya tanto la suma de los costes de enlace como la suma de los pesos de los vértices con grado superior a dos [15]. Se trata de un concepto central en teoría de la complejidad que sirve para clasificar aquellos problemas de decisión cuya solución, aunque fácil de verificar una vez encontrada, parece no poder encontrarse mediante algoritmos que crezcan únicamente de forma polinómica con el tamaño de la entrada.

Un problema se dice NP-Completeness cuando cumple dos condiciones: primero, cualquier solución propuesta puede comprobarse en tiempo razonable y, en segundo lugar, es tan complejo como el resto de los problemas de la clase NP, pues todos ellos pueden transformarse en él mediante un procedimiento eficiente.

Cuando un problema es de tipo NP-completeness, la estrategia práctica pasa por recurrir a aproximaciones, heurísticas o métodos probabilísticos que, sin garantizar la solución óptima, permiten obtener respuestas muy buenas en tiempos de cálculo aceptables.

En este proyecto se exploran varias estrategias innovadoras, incluyendo desde adaptaciones directas de Prim hasta un método genético que combinen diversas configuraciones. Para ello, se desarrollarán algoritmos de optimización de rutas que, a diferencia de los métodos clásicos de MST, incluyan la penalización por switches.

1.2. PROBLEMAS EN EL DISEÑO DE REDES

El diseño de redes de telecomunicaciones conlleva una serie de retos técnicos y operativos que trascienden a la minimización de distancias o latencias, y se adentran en la necesidad de gestionar estructuras cada vez más heterogéneas y dinámicas.

En primer lugar, la escalabilidad de la red supone un desafío esencial. A medida que crece el número de nodos, tanto routers como switches intermedios, y la demanda de tráfico fluctúa, se debe garantizar que la infraestructura sea capaz de soportar incrementos exponenciales de usuarios y datos sin que se produzcan cuellos de botella o degradaciones bruscas en el rendimiento. Históricamente, la incorporación de nuevos puntos de presencia o CPDs obligaba a redefinir rutas enteras, replantear esquemas de direccionamiento IP y actualizar protocolos de enrutamiento, con direcciones de prefijo agregadas de forma manual [16]. Sin embargo, en entornos modernos con cientos de miles de prefijos y cambiantes patrones de tráfico, mantener una red escalable exige automatización en la agregación de rutas, redistribución de cargas y ajustes continuos en el plan de direccionamiento, de manera que la adición de un nuevo nodo no perturbe la convergencia de los protocolos de enrutamiento interior (IGP) y exterior (EGP) [17]. A modo de ejemplo de esto, en la Figura 7 se puede apreciar el camino de ida que recorre una petición de DNS desde la red de Aire Networks hacia la red de Cloudflare, junto con las latencias detectadas en cada nodo intermedio.

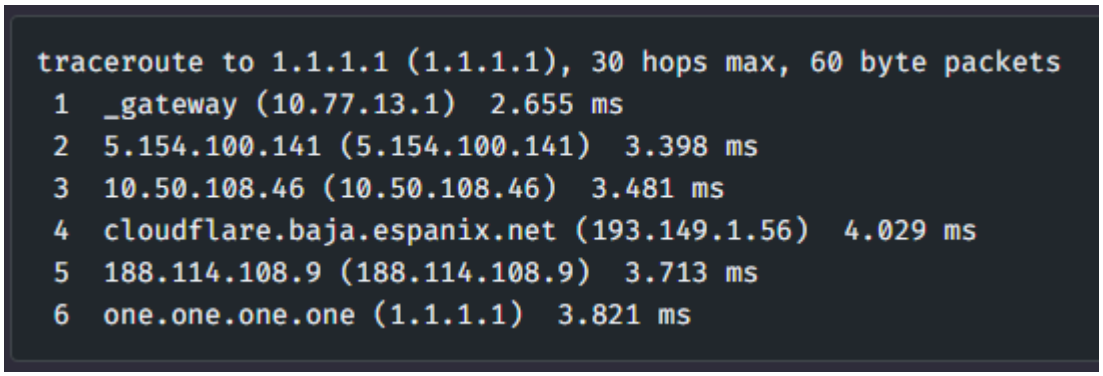


Figura 7. Tracert desde la red de Aire Networks hacia las DNS de Cloudflare.

Extraído de: https://lg.as29119.net/

En segundo lugar, la gestión de la heterogeneidad tecnológica incide directamente en la complejidad del diseño. En una red coexisten enlaces de alta capacidad en la parte troncal (por ejemplo, DWDM sobre fibra óptica), conexiones de acceso fijo (GPON, xDSL) o enlaces inalámbricos (4G/5G, satélite). Cada tecnología exhibe características propias de latencia, variabilidad de pérdida de paquetes y perfiles de tráfico. Esta diversidad obliga a definir métricas de coste que reflejen no solo la distancia física o el ancho de banda, sino también la calidad de transmisión y la probabilidad de congestión [17].

En tercer lugar, otro problema para tener en cuenta es la resiliencia y la convergencia de protocolos. Cuando un enlace o nodo falla, las rutas de respaldo deben activarse con la mayor celeridad posible para evitar interrupciones de servicio. Por ejemplo, en redes que emplean BGP juntamente con IGP, la convergencia ante fallos puede tardar del orden de segundos si no se aplican soluciones basadas en la configuración de rutas de contingencia [18].

En cuarto lugar, la variabilidad de la carga es otro factor que complica el dimensionamiento. En momentos de picos de tráfico, la demanda puede multiplicarse en un corto intervalo de tiempo. En el trabajo “Practical network support for ISP traffic engineering,” [19] describen cómo la ingeniería de tráfico proactiva debe balancear la carga entre múltiples caminos para evitar la saturación de enlaces específicos. Sin embargo, se distribuye el tráfico de forma uniforme sólo cuando las rutas tienen el mismo coste, y no contempla variaciones en el perfil de tráfico. En una red real es necesario combinarlo con elementos de monitorización en tiempo real que adapten dinámicamente las rutas en función del uso efectivo de los enlaces.

En cuanto al plan de redundancia, si se realiza un diseño excesivamente redundante puede suponer topologías sobredimensionadas, donde múltiples enlaces alternativos disponen de los mismos recursos, aumentando la complejidad de gestión y el riesgo de bucles ante cualquier fallo de configuración. En contraste, un diseño con redundancia insuficiente expone la red a interrupciones prolongadas en caso de fallo. Esto puede solucionarse con el uso de árboles de recubrimiento disjuntos (disjoint

spanning trees) que compartan muy pocos nodos críticos, de modo que si un switch central falla, exista una ruta de respaldo que no dependa de él [21].

El control de bucles en capa dos y la convergencia de la capa tres son problemas que están relacionados. En entornos tradicionales, se utiliza el protocolo STP (Spanning Tree Protocol) para evitar bucles, pero inhabilita enlaces redundantes, reduciendo la capacidad de la red. En redes con dispositivos compatibles con Rapid STP (RSTP) o Multiple STP (MSTP), la reconvergencia puede tardar entre uno y tres segundos, lo que genera interrupciones momentáneas debido al reencaminamiento de tráfico por una vía redundada [18]. Además, cuando se combina STP con enrutamiento IP, se corre el riesgo de inconsistencias temporales entre la topología activa en capa dos y las rutas calculadas en capa tres, provocando pérdida de conectividad hasta que ambos planos converjan ya que la tabla de rutas tendría que rehacerse en los nuevos puntos de enrutamiento.

En quinto lugar, otro tema importante es la limitación de capacidad ascendente y descendente en nodos de acceso y agregación. Como es normal, el ancho de banda de los accesos residenciales o empresariales es mucho menor que el de los enlaces troncalizados. Por ello, se debe planificar cuidadosamente el dimensionamiento de enlaces ascendentes (uplinks) en switches de agregación para evitar que los usuarios saturen los puertos compartidos en determinados momentos del día [20].

Por último, la interoperabilidad y la gestión de políticas en entornos multi-proveedor añaden una capa de complejidad. Los ISP suelen interconectarse mediante acuerdos de peering y tránsito que involucran múltiples operadores. La falta de alineación en las políticas entre proveedores puede derivar en degradaciones en el servicio cuando el tráfico atraviesa múltiples redes saturadas y el tráfico del ISP no se prioriza por la política acordada [17].

1.3. MÉTODOS PARA EL DISEÑO DE REDES

En el ámbito del diseño de las redes de telecomunicaciones, los métodos utilizados han evolucionado desde enfoques puramente teóricos basados en grafos hasta técnicas computacionales avanzadas que combinan optimización matemática, heurísticas específicas y metaheurísticas adaptadas a las particularidades de infraestructuras ISP. A continuación, se describen los principales métodos empleados, profundizando en las soluciones algorítmicas y metodologías de modelado.

Los métodos exactos para el diseño de redes se fundamentan en la formulación del problema como un modelo de programación matemática, normalmente de tipo entero o mixto-entero (MIP). En estas formulaciones, cada arista y cada vértice puede representarse mediante variables binarias que indican si el enlace está presente en la topología final o si el nodo funcionará como punto de conmutación. Así, el objetivo suele expresarse como la minimización de una función lineal que combine el coste de los enlaces y el coste de los nodos cuando su grado excede un umbral específico. Un trabajo clásico en este sentido describe un modelo de MIP en el que se agregan restricciones de flujo para asegurar la conectividad y desigualdades de grado para penalizar nodos de alto grado [22]. Sin embargo, la resolución exacta de estos modelos se vuelve inviable para redes de más de unas pocas decenas de nodos, debido a la suma de variables y restricciones [23].

Para evitar esta limitación, se han desarrollado técnicas de descomposición y relajación. El método de descomposición de Dantzig–Wolfe [24] convierte el problema original en un subproblema maestro y un subproblema de rutas: el maestro decide qué subconjuntos de enlaces incorporar, mientras que el subproblema genera rutas factibles que se aportan al maestro para mejorar la solución iterativamente. Otra estrategia es la relajación lagrangiana, en la que se relajan las restricciones de grado y se penalizan mediante multiplicadores de Lagrange [25], obteniendo un problema de flujo mínimo que se resuelve de forma más eficiente. La relajación permite obtener cotas inferiores muy ajustadas que guían búsquedas más efectivas. Estas variantes exactas, aunque sofisticadas, presentan como principal inconveniente la necesidad de

tiempos de cómputo crecientes de forma exponencial al aumentar el tamaño del grafo, por lo que su uso queda restringido a casos de estudio de pequeña o mediana escala.

Una variante extendida de Prim introduce un término adicional en el coste de cada nodo cuando su grado alcanza un umbral crítico por encima de dos conexiones [26], lo cual incita la construcción hacia topologías más repartidas. Del mismo modo, existe otra versión en la que, en vez de optar siempre por la menor arista, evalúa para cada arista candidata la suma de su coste de enlace más el coste asociado a convertir el nodo de origen o destino en switch de mayor grado; esta heurística selecciona la arista de mínimo coste modificado en cada paso, sin retrocesos [27]. Aunque estas estrategias no garantizan óptimos globales, su simplicidad permite tiempos de ejecución casi lineales, por lo que resultan adecuadas para grafos con centenas o miles de nodos, especialmente cuando se usan como punto de partida para refinamientos posteriores.

Por otra parte, los métodos de búsqueda local han demostrado un rendimiento notable. En este enfoque, iniciando a partir de una solución factible [28], por ejemplo, un árbol de expansión mínima construido con coste de enlaces, se exploran vecinos mediante operaciones basadas en cambiar una arista por otra que preserve la conectividad o reubicar una conexión de un nodo a otro, tratando de mejorar la función objetivo que combina costes de enlaces y penalizaciones de nodo. Al iterar estas modificaciones locales, se puede escapar de óptimos locales mediante criterios de aceptación que permitan, en ocasiones, incrementar el coste temporalmente o registrar las mejores soluciones encontradas en un mecanismo que impida revisitar configuraciones recientes [29]. Estas estrategias híbridas combinan la rapidez de una heurística constructiva inicial con la capacidad exploratoria de la búsqueda local.

Los algoritmos genéticos (AG) han sido ampliamente aplicados al problema de árboles de recubrimiento con penalización de vértices [30]. En este contexto, cada individuo de la población representa un conjunto de rutas parciales que luego se recomponen mediante operadores de cruce y mutación. Por ejemplo, un cromosoma puede codificarse como un vector de bits que indica, para cada arista, si se incluye en la topología; el operador de cruce mezcla subconjuntos de aristas de dos padres [31].

Otros AG emplean representaciones basadas en lista de adyacencia, donde cada gen define una rama de conexión; el cruce se realiza intercambiando subárboles completos [32]. Tras el cruce, la mutación puede consistir en la adición o eliminación aleatoria de aristas o en la modificación de la asignación de grado de algunos nodos, buscando diversificar la población.

Las técnicas de enjambre de partículas (PSO) también se han adaptado al dominio de diseño de redes, aunque su aplicación es menos intuitiva que para problemas continuos. En estos modelos, cada partícula codifica una posible estructura topológica, representada mediante vectores que se traducen a conjuntos de aristas según un mecanismo de umbralización; la velocidad de cada partícula se actualiza en función de la mejor solución individual y de la mejor solución global, incentivando la convergencia hacia un óptimo [33]. Algunas investigaciones extienden PSO combinándolo con una fase de refinamiento de búsqueda local, de modo que, una vez que la partícula converge a una región prometedora del espacio de soluciones, se depura la topología resultante [37].

Por último, las técnicas multi objetivo han cobrado importancia frente a la tradicional minimización mono objetivo, dado que el diseño de redes de telecomunicaciones frecuentemente debe equilibrar la reducción de costes con la maximización de la resiliencia o la calidad de servicio. En este marco, el problema se formula con dos o más funciones objetivo concurrentes y se busca obtener el conjunto de topologías que no pueden mejorar en un criterio sin empeorar en otro. Algoritmos como NSGA-II (Non-Dominated Sorting Genetic Algorithm II) han sido adaptados a la codificación de redes, donde la población evoluciona seleccionando individuos con base en la dominancia y la diversidad de soluciones [35].

1.4. PROBLEMA DE LOCALIZACIÓN DE SWITCHES

El problema de la localización de switches en una red de telecomunicaciones se basa en determinar qué nodos deben equiparse con este tipo de dispositivos para garantizar conectividad, resiliencia y eficiencia operativa, sin incurrir en instalaciones redundantes o innecesarias. A diferencia de la tarea genérica de seleccionar rutas óptimas entre nodos, se debe decidir primero qué y cuántos switches desplegar, cada uno con un coste y una capacidad asociados, y luego diseñar la topología de enlaces que interconecte estos equipos con los routers de acceso y otros dispositivos de red. Este planteamiento remite a problemas de localización clásica [36] adaptado al dominio de redes, donde las ubicaciones deben examinarse para cubrir demanda de usuarios y puntos de intercambio con el menor gasto total.

A menudo se añade la restricción de que cada switch tenga un grado máximo o mínimo, ya que el dispositivo sólo soporta un número finito de puertos [37]. Además, en muchos casos se requiere que cada nodo de usuario o router de acceso se conecte a algún switch, modelo que comparte similitudes con el "Uncapacitated Facility Location Problem" en teoría de la optimización combinatoria [38], un problema el cual dado un conjunto de posibles ubicaciones de instalaciones y conjunto de clientes, se debe decidir qué instalaciones abrir y cómo asignar cada cliente a una instalación de modo que se minimice la suma de los costes de apertura y de asignación, sin límite en la capacidad de las instalaciones.

Una de las variantes más relevantes para redes ISP es el problema de localización de switches con restricciones de cobertura geográfica. En ella, la posición física del switch influye en la latencia esperada y en la longitud total de cableado. En este contexto, la función objetivo no solo incluye el coste fijo de instalar un switch, sino también un término proporcional a la distancia entre cada cliente o nodo de entrada de tráfico y el switch de capa tres más próximo [39]. Este enfoque se asemeja al "p-Median Problem" en planificación de servicios, donde los switches funcionan como centros que sirven a un conjunto de clientes; sin embargo, en el ámbito de telecomunicaciones, el peso de cada cliente suele depender del volumen de tráfico o de la criticidad del servicio [40].

En el caso concreto de un backbone, los switches se sitúan en nodos estratégicos como puntos de intercambio de Internet y centros de datos. El problema de decidir en cuáles de estos puntos desplegar switches de alta densidad no es sencillo de abordar, ya que cada uno soporta diferentes niveles de demanda agregada y posee costes heterogéneos. Un switch en un IXP principal conlleva un sobrecoste, pero puede atender gran parte del tránsito de datos. Estudios previos en este ámbito han demostrado que ignorar este efecto puede conducir a soluciones subóptimas que sobresaturan puntos críticos y generan cuellos de botella, incluso cuando la suma de costes de enlace es baja [41].

Otra dimensión relevante es la resiliencia ante fallos. Cuando se decide la localización de switches, debe garantizarse que, ante el fallo de un switch o enlace, la topología todavía cumpla con requisitos mínimos de cobertura. Esto se modela añadiendo restricciones de redundancia; cada router de acceso debe conectarse a, al menos, dos switches distintos o disponerse rutas alternativas que no compartan ningún switch ni enlace crítico. De esta manera, el problema se acerca al "Redundant Facility Location Problem" [42] adaptado a grafos, en el que se exige que cada cliente tenga una cobertura múltiple y se minimiza el coste total multiplicado por factores de tolerancia al fallo.

En términos de algoritmos, las formulaciones MIP que describen el problema de localización de switches son muy similares a las de localización de instalaciones en optimización general, menos en la inclusión de variables de asignación que vinculan cada router o equipo de borde a un switch concreto. La desventaja principal radica en que, para grafos de tamaño medio, la resolución exacta se vuelve inabordable [44]. Por ello, se suelen adoptar soluciones heurísticas de relajación Lagrangiana que relajan las restricciones de asignación de clientes a switches, permitiendo descomponer el problema en subproblemas independientes de cobertura [45]. Alternativamente, los métodos de partición y corte (cutting planes) se utilizan para generar iterativamente cortes que impiden soluciones inconexas [46].

En la práctica, entre las heurísticas constructivas para la localización de switches destaca el enfoque de "incremental hub placement", donde se añaden switches uno a

uno en las ubicaciones que ofrecen mayor reducción del coste marginal, calculado en función de la mejora de cobertura y la disminución de costes de enlace para los routers adyacentes [47]. Posteriormente, se lleva a cabo una fase de ajuste local donde, si se retira un switch de bajo rendimiento, sus clientes se reasignan buscando reducir el coste total [48]. Estas heurísticas, si bien rápidas, pueden quedarse atrapadas en óptimos locales, por lo que a menudo se combinan con otras técnicas para explorar configuraciones que impliquen mover o intercambiar switches entre ubicaciones [49].

Asimismo, se han desarrollado métodos metaheurísticos multiobjetivo para equilibrar simultáneamente el coste de despliegue de switch y la latencia total de los clientes finales. En estos métodos, cada individuo codifica un conjunto de switches y la reasignación de cada router a uno de ellos; emplean operadores genéticos que cruzan subconjuntos de ubicaciones y ajustan poblaciones según métricas de dominancia [50]. Estas técnicas permiten disponer de un frente de soluciones: algunas con menor número de switches, pero mayor latencia, y otras con mayor cantidad de switches ubicados más cerca de los clientes, reduciendo la latencia a costa de desplegar equipamiento adicional [51].

Al incorporar estos métodos, es fundamental tener en cuenta que los switches no solamente agregan un coste fijo, sino que su capacidad de conmutación limita el número de interfaces que pueden atender simultáneamente. Cuando la demanda global supera la capacidad de todos los switches seleccionados, se hace necesaria la incorporación de enlaces conmutados de respaldo o la creación de clústers de switches interconectados, complicando la formulación y ampliando el espacio de soluciones.

Finalmente, cabe mencionar la importancia de datos empíricos y herramientas de simulación para validar los modelos de localización de switches. A partir de mapas de topología real, se extraen subgrafos representativos que simulan la red de un ISP en una región concreta. Sobre estos subgrafos, se evalúan diferentes configuraciones de localización utilizando simuladores de tráfico que inyectan modelos de demanda basados en estadísticas reales [52].

1.5. ESTADO DEL ARTE

En este apartado se recoge en detalle el estado del arte en torno a los switches, su evolución tecnológica, tipos, funciones, desempeño y relevancia dentro de las infraestructuras ISP, con especial atención al estudio “Modelo de localización de switches con penalización de grado variable” [43] como ejemplo de trabajo reciente que aborda la problemática de determinar la ubicación y características óptimas de estos dispositivos en entornos de gran escala.

Un conmutador, interruptor o switch es un dispositivo fundamental en la arquitectura de cualquier red de telecomunicaciones, pues actúa como punto neurálgico que interconecta segmentos de red y dirige el tráfico de datos. Si nos ceñimos a su concepción más elemental, es un dispositivo electrónico que recibe tramas de datos en uno de sus puertos y las reenvía únicamente hacia el puerto de destino correspondiente, de acuerdo con la dirección MAC (Media Access Control) de la trama. Esta operación de “switching” permite segmentar dominios de colisión, reduciendo la congestión en la red y mejorando significativamente el rendimiento respecto a concentradores tradicionales. Al nivel de la capa dos del modelo OSI, el switch construye dinámicamente una tabla de direcciones MAC asociadas a puertos físicos mediante un proceso de aprendizaje. Cada trama que recibe le permite registrar la asociación MAC – puerto, de modo que futuras tramas se entreguen de forma directa [53].

Desde la aparición de los primeros switches Ethernet de 10 Mbps en la década de 1990, la industria ha recorrido un largo camino hacia los dispositivos de hoy en día. A principios de los 2000, la consolidación de Fast Ethernet (100 Mbps) y Gigabit Ethernet (1 Gbps) en infraestructura de acceso impulsó la adopción de switches con ASICs dedicados para forwarding en hardware, lo que redujo drásticamente la latencia de reenvío a microsegundos [59]. Posteriormente, la llegada de 10 Gbps y 40 Gbps en el backbone local requirió switches capaces de gestionar flujos de datos a velocidades de línea completa, implementando memorias TCAM (Ternary Content-Addressable Memory) para búsquedas rápidas de direcciones MAC y reglas de ACLs [60].

En la década de 2010, el despliegue masivo de tendidos de fibra con velocidades de 100 Gbps y la transición hacia arquitecturas de centro de datos definidas por software impulsaron el desarrollo de switches con soporte nativo para OpenFlow y tablas de flujo programables. Estos dispositivos permiten a los controladores SDN instalar rutas de forwarding de manera dinámica y centralizada, ofreciendo flexibilidad para implementar políticas de enrutamiento específicas según la demanda [61]. Además, la aparición de P4 (Programming Protocol-independent Packet Processors) abrió la posibilidad de definir protocolos de procesamiento de paquetes a medida, transformando los switches en elementos de red altamente configurables que se adaptan a protocolos emergentes sin necesidad de reemplazar hardware [62].

En la actualidad, la mayoría de los switches desplegados en núcleos de ISP combinan capacidades de conmutación y enrutamiento, gestionando tanto el reenvío basado en direcciones MAC como la toma de decisiones en función de cabeceras IP y parámetros de priorización de tráfico [54]. Esta convergencia de funciones ha dado lugar a una nueva familia de dispositivos híbridos que realizan procesamiento de tramas a velocidades muy altas, minimizando la latencia interna y facilitando la implementación de políticas de enrutamiento IP [55].

En el ámbito de los proveedores de servicios de Internet, los switches suelen clasificarse en función de su rol y capacidad dentro de la jerarquía de la red. A grandes rasgos, se distinguen tres niveles principales [56]:

1. Switch de acceso (Edge Switch): Ubicado en la frontera entre el cliente final y la red ISP. Sus funciones se limitan, por lo general, a proporcionar conectividad a usuarios y clientes, aplicar políticas básicas de control de acceso (ACLs) y clasificar tráfico. Estos switches tienen un número elevado de puertos de acceso de baja velocidad (entre 1 Gbps y 10 Gbps) y suelen integrar módulos PoE (Power over Ethernet) para alimentar dispositivos terminales.
2. Switch de distribución (Aggregation Switch): Opera en el nivel intermedio, consolidando tráfico que proviene de múltiples switches de acceso. En esta capa se ubican dispositivos con mayor capacidad de conmutación (10 Gbps, 40 Gbps o 100 Gbps por puerto) y funciones avanzadas de capa dos y capa tres, como VLAN

trunking, MPLS (Multiprotocol Label Switching) y balanceo de carga. Los switches de distribución gestionan la agregación de tráfico y suelen interconectar varios switches de acceso con el backbone troncal [57].

3. Switch de núcleo (Core Switch): Se encargan de enrutar grandes volúmenes de tráfico entre centros de datos, puntos de intercambio de Internet (IXPs) o enlaces troncales de larga distancia. Estos switches soportan puertos de 100 Gbps o incluso 400 Gbps y están optimizados para alta densidad de puertos y baja latencia. Su arquitectura interna se diseña para minimizar el retardo de conmutación, habilitando forwarding en hardware mediante ASICs (Application-Specific Integrated Circuits) o NPUs (Network Processor Units) [58].

Durante los últimos cinco años, se ha prestado especial atención al problema de dimensionamiento de switches y su localización en infraestructuras ISP de gran tamaño. Utilizando un enfoque de relocalización dinámica de switches en función de cambios de demanda predictiva [43], se define en este trabajo que cada switch asume un coste fijo al instalarse, más un término adicional que crece linealmente con su grado, reflejando la mayor complejidad de gestión y la menor eficiencia interna de switches muy cargados. Esto mezcla variables binarias para la decisión de instalación y variables continuas de flujo para la asignación de enlaces, introduciendo restricciones de grados máximos que evitan saturar físicamente los puertos.

El problema MBV surge en contextos en los que, más allá de minimizar la suma de pesos de los enlaces, interesa reducir la cantidad de nodos que ramifican en más de dos conexiones. Carrabs et al. [64] identificaron la necesidad de contar con conjuntos de grafos que fuesen lo suficientemente repartidos para que el árbol generador óptimo incluyera un número de ramificaciones no trivial, de modo que los algoritmos exactos y heurísticos pudieran probarse en situaciones realistas donde la reducción de switches resultara relevante.

En concreto, estos autores pusieron a disposición pública un paquete de instancias aleatorias de grafos conectados en los que se varía sistemáticamente el número de vértices y la densidad de aristas, generando múltiples réplicas de cada tamaño para garantizar robustez estadística en los experimentos. De este modo, se consiguió un

marco de evaluación coherente que permitió comparar con objetividad el rendimiento de diferentes metodologías.

1.6. OBJETIVOS DEL PROYECTO

Los objetivos de este trabajo se enmarcan en la búsqueda de una optimización tangible de las redes de telecomunicaciones de los ISP mediante la reducción del número de dispositivos que actúan como nodos de conmutación o switches. Se pretende demostrar que una adecuada selección y configuración de dichos nodos puede mantener la integridad, la eficiencia y la resiliencia de la red, a la vez que disminuye el coste operativo y la complejidad de la infraestructura.

El objetivo principal consiste en lograr que, para cualquier topología de red dada, se obtenga una configuración en la que el número de nodos con más de dos conexiones y el coste de los enlaces que conforman el árbol sea mínimo. Para ello, se tendrá en cuenta que no es imprescindible equipar cada cruce de rutas con un switch de alta capacidad, sino que, mediante un diseño adecuado, un número reducido de estos dispositivos puede atender todo el tráfico.

1.7. ESTRUCTURA DEL DOCUMENTO

En el Capítulo 1 (“Introducción”) se establecerá el marco general de la investigación. Se inicia con la contextualización y motivación del estudio, destacando la importancia de optimizar la topología de redes de los ISP mediante la reducción de switches. A continuación, se identifican y analizan los problemas más relevantes en el diseño de redes contemporáneas, prestando especial atención a la complejidad asociada con nodos de conmutación. Se detalla los métodos clásicos y heurísticos existentes para el diseño de redes, mientras que se habla de forma formal sobre el problema de localización de switches y sus particularidades para un proveedor de servicios de Internet. El estado del arte profundiza en definiciones, evoluciones tecnológicas y enfoques recientes relacionados exclusivamente con los switches en arquitecturas ISP. Finalmente, se fijan los objetivos concretos del proyecto y se describe la estructura del documento, explicando el propósito de cada capítulo.

El Capítulo 2 (“Material y métodos”) describe los elementos y procedimientos empleados para llevar a cabo la investigación. Se presenta el modelo matemático base de este proyecto, el análisis previo de topologías de red y datos empleados para el experimento; se expone la metodología experimental, con el diseño de algoritmos, criterios y proceso de validación. Seguidamente introduce de manera general cada uno de los métodos y algoritmos a desarrollar, para posteriormente recoger con detalle el desarrollo paso a paso de cada algoritmo: en primer lugar, el algoritmo de Prim clásico modificado para contabilizar el coste de nodos, después la versión dinámica de Prim, y a continuación las tres heurísticas N1, N2 y N3 (2.6, 2.7 y 2.8). Finalmente, se presenta el diseño y la implementación del método genético.

El Capítulo 3 (“Resultados y discusión”) se ocupa de mostrar y analizar los datos obtenidos. En primer lugar, se describen los datos experimentales empleados y se puntualizan las herramientas de programación y análisis. Posteriormente, se presenta de forma ordenada los resultados obtenidos por cada algoritmo y se realiza un análisis comparativo entre las diversas técnicas, comparando directamente la suma de costes en la reducción de switches alcanzada junto con el MST obtenido en cada caso.

Finalmente se comentan las ventajas y limitaciones de cada estrategia a partir de los resultados, y se valida la robustez de la metodología.

En el Capítulo 4 (“Conclusiones”) se sintetizan las aportaciones del estudio. Se aporta un resumen los resultados más relevantes obtenidos, destacando los algoritmos capaces de lograr la mayor reducción de switches sin degradar el rendimiento. Se comentan las conclusiones generales y se reflexiona sobre el cumplimiento de los objetivos planteados. Finalmente, plantea líneas de trabajo futuro y propuestas de mejora, apuntando posibles extensiones del proyecto.

El Capítulo 5 (“Anexos”) reúne material suplementario que complementa al resto del trabajo. Se incluyen diagramas de flujo y esquemas de los algoritmos desarrollados, así como las tablas de resultados de todos los archivos analizados.

Por último, el Capítulo 6 (“Bibliografía”) enumera todas las referencias consultadas y citadas a lo largo de todo el trabajo.

2. MATERIAL Y MÉTODOS

2.1. MODELO MATEMÁTICO

El Switch Location Problem (SLP) consiste en encontrar el subgrafo de entrega de coste total mínimo (suma de costes de instalación y asignación) como se aprecia en la Figura 8. Se considera una red de transbordo representada por un grafo no dirigido $G = (V, E)$ y $S \subseteq V$ el subconjunto de nodos candidatos a switch (grado ≥ 3). Un subgrafo de entrega es un subgrafo conectado que contiene todos los vértices de G. Aquellos nodos de grado ≥ 3 en el subgrafo de entrega deben equiparse con un switch, lo que implica:

Un coste de instalación f_j si el switch se ubica en el nodo $j \in S$.

Un coste de asignación g_j cada vez que incide una arista $j \in S$ a dicho switch.

Una variable entera y_j que cuenta cuántas aristas incidentes en j pagan coste por asignación.

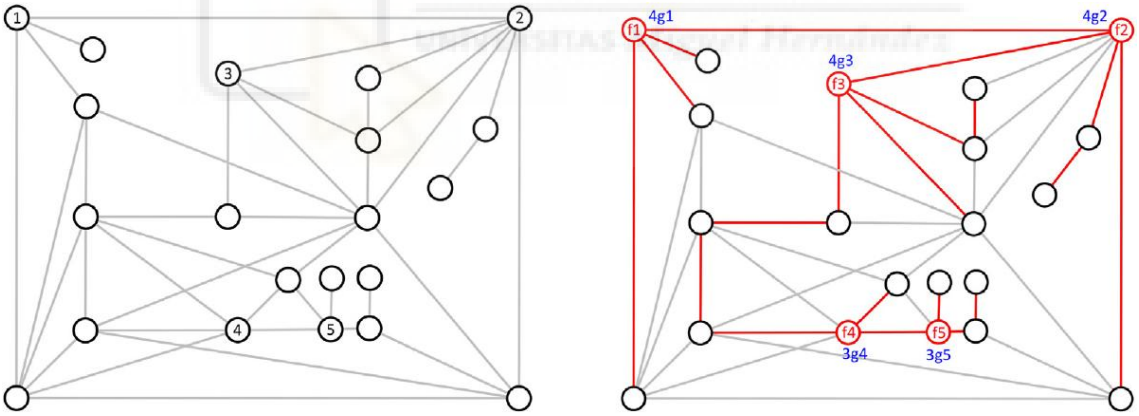


Figura 8. Ejemplo de ubicación y asignación de switches.

Extraído de: Locating switches. Expert systems with applications, 136, 338-352.

El modelo principal $SLP\ (G, f, g)$ es:

$$\min \sum_{j \in S} (f_j q_j + g_j y_j)$$

Definimos variables binarias:

$$q_j = \begin{cases} 1\ si\ hay\ switch\ en\ j \\ 0\ si\ no\ switch\ en\ j \end{cases}, x_{uv} = \begin{cases} 1\ \ si\ el\ árbol\ incluye\ arista\ uv \in E \\ 0\ si\ el\ árbol\ no\ incluye\ arista. \end{cases}$$

Restricciones esenciales para generar un árbol a partir de un grafo inicial:

1. Conexo y acíclico: $\sum_{uv \in E} x_{uv} = |V| - 1, \sum_{v:uv \in E} x_{uv} \geq 1\ \forall u \in V$, y no hay ciclos: para todo $U \subset V, \sum_{uv \in U} x_{uv} \leq |U| - 1$
2. Grado y switches: $\sum_{v:jv \in E} x_{jv} \leq 2 + 2q_j\ \ \forall j \in S$, de modo que si no se instala switch ($q_j = 0$), j tenga grado ≤ 2 y si $q_j = 1$ pueda llegar a su grado original.
3. Asignación de costes: $y_j \geq \sum_{v:jv \in E} x_{jv} - 2q_j\ \forall j \in S$, de modo que y_j cuente las aristas extra sobre las dos primeras.

Tras definir las variables y restricciones, garantizamos que la solución sea un árbol que conecte todos los nodos y que los vértices de grado ≥ 3 sean tratados como un switch, computando el coste de asignación g_j.

2.2. APORTACIONES Y ALGORITMOS HEURÍSTICOS

En esta sección se presenta una visión integradora de los algoritmos que sustentan este proyecto, cuyo propósito central consiste en reducir el coste de los enlaces junto con el número de switches en redes modeladas como grafos ponderados. Para ello, partimos de las formulaciones clásicas de optimización de árboles de expansión mínima y las transformamos, paso a paso, en versiones adaptadas, heurísticas y metaheurísticas. Se describe su esencia, los mecanismos que incorporan para penalizar la aparición de switches y la forma en que cada variante extiende o modifica a su predecesor en la cadena de desarrollo.

El punto de partida es el algoritmo de Prim, ampliamente conocido en teoría de grafos. En su versión básica, Prim construye de manera recursiva un árbol que cubre todos los nodos minimizando la suma de los pesos de las aristas elegidas. Sin embargo, en esa formulación original no existe noción del coste asociado al vértice. Cada arista pesa lo mismo aunque en la red final, si un nodo termina con más de dos conexiones, implicará la instalación de un switch, con el consiguiente recargo operativo. Por ello, la primera adaptación consiste en incorporar de modo directo un coste que penalice aquellas aristas cuya inclusión elevaría el grado de un vértice por encima de dos.

A partir de este concepto original y novedoso, se han diseñado cinco algoritmos innovadores orientados a unificar la red reduciendo al máximo los costes asociados a nodos y switches. Estas variantes abarcan desde un recargo estático hasta métodos que ajustan progresivamente ese sobrecoste de manera proporcional o mediante particiones aleatorias, y culminan en un enfoque evolutivo que combina de forma iterativa múltiples soluciones parciales para lograr una topología óptima.

- Prim clásico con coste de nodos
- Prim dinámico
- Heurístico N1
- Heurístico N2
- Heurístico N3
- Método Genético

2.3. PRIM CLÁSICO CON COSTE DE NODOS

En este apartado se describe la forma en que se adapta el algoritmo clásico de Prim para incorporar el coste asociado a aquellos nodos que asumen más de dos conexiones y, por tanto, se identifican como switches. El objetivo es que la construcción del árbol de expansión mínima no se limite únicamente a minimizar el peso de los enlaces, sino que también penalice el uso excesivo de un mismo nodo como punto de concentración de conexiones.

Pseudocódigo del Algoritmo 1 – Prim clásico con coste de nodos
Input: directorio_entradas, directorio_salidas
For each archivo in directory_entradas whose name contains "_MATRICES":
1. Leer matriz_costes y vector_pesos desde archivo
2. Ejecutar Prim sobre matriz_costes → arbol, coste_aristas
3. Contar grado de cada nodo en arbol → nodos_conectados (grado ≥ 3)
4. coste_nodos ← suma de vector_pesos para cada nodo en nodos_conectados
5. suma_total ← coste_aristas + coste_nodos
6. nombre_salida ← sustituir "_MATRICES" por "_CLASICO" en el nombre de archivo
7. Escribir en nombre_salida: lista de aristas, nodos_conectados, coste_aristas, coste_nodos, suma_total
End of Pseudocódigo del Algoritmo 1

Para llevarlo a cabo, se parte de la representación habitual de la red mediante una matriz de costes, en la cual cada celda contiene el peso de transmisión o el coste de instalación de un enlace entre dos nodos. Adicionalmente, se dispondrá de un vector paralelo que asigna a cada nodo un valor fijo que representa su coste de switch en cuanto supera la segunda conexión.

En la implementación, se sigue el esquema básico de Prim, pero con un matiz fundamental. Se comienza por seleccionar de forma arbitraria un nodo de partida y se marca como visitado. Paralelamente, se mantiene un contador de grado para cada nodo, inicialmente a cero, que irá registrando cuántas conexiones ha adquirido en el árbol en construcción. Posteriormente se busca, entre todos los enlaces que comunican un nodo ya incorporado al árbol con otro aún no visitado, aquel con menor coste asociado. Sin embargo, a diferencia de Prim tradicional, cuando un enlace parte de un nodo que ya cuenta con exactamente dos conexiones en el árbol, se suma al peso habitual de ese enlace la penalización fija asignada a dicho nodo. De este modo, en el momento en que un vértice alcanza su tercera conexión, el peso del enlace que lo habilita como switch aparece incrementado por el coste del propio conmutador.

A partir de la cuarta conexión de un mismo nodo, no se vuelve a añadir ninguna penalización adicional, pues se considera que el coste de switch ya se ha asumido en el momento de su tercera conexión.

2.4. PRIM DINÁMICO

En este apartado se describe cómo se modifica el algoritmo clásico de Prim para que, durante su ejecución, se tenga en cuenta de inmediato el coste adicional que supone que un nodo alcance tres conexiones y se asuma un switch. La idea es que, cada vez que se evalúa cualquier posible enlace que conecte un nodo ya incluido en el árbol con otro todavía sin incorporar, se ajuste el peso de ese enlace en función de si el nodo de origen ya tiene dos conexiones; en ese caso, ese enlace recibe un sobre coste equivalente al coste asignado a ese nodo, de modo que la decisión de incorporarlo se basa en un coste total que combina el enlace original más dicho coste.

Pseudocódigo del Algoritmo 2 – Prim Dinámico

```
Input:
 • dir_in  ← carpeta con archivos "*_RED.txt"
 • dir_out ← carpeta destino para resultados
Para cada fichero f en dir_in que termine en «_RED.txt»
 1. (C, P) ← leer_matrices(f)
 2. (T, cost_arcs, deg) ← PRIM_DINÁMICO(C, P)
 3. S ← {u | deg[u] ≥ 3}
 4. cost_nodes ← ∑_{u∈S} P[u]
 5. total ← cost_arcs + cost_nodes
 6. g ← replace(f, "_RED", "_DINAMICO")
 7. guardar(T, cost_arcs, S, cost_nodes, total, dir_out/g)
End For

Function PRIM_DINÁMICO(C, P):
  Inicializa visited[1]=True, deg[]=0, T←[], cost_arcs←0
  While |T| < n–1 do
    Find edge (i,j) with visited[i] and ¬visited[j] minimiza C[i,j] + (P[i] if deg[i]=2 else 0)
    Mark visited[j]=True; deg[i]++, deg[j]++
    Append (i,j, original, extra) to T; cost_arcs += original
  End While
  Return (T, cost_arcs, deg)
End of Pseudocódigo del Algoritmo 2
```

El algoritmo comienza escogiendo de forma arbitraria un nodo inicial y marcándolo como parte del árbol. A partir de ahí, se mantiene un registro del número de enlaces que ya tiene cada vértice dentro de la estructura en crecimiento. Originalmente, todos los grados están en cero. A cada paso, se consideran todas las posibles conexiones que parten de los vértices ya incluidos y llegan a aquellos que aún no lo están. Para cada una de esas líneas, se toma el coste original. Si el nodo de partida tiene cero o una conexión previa, dicho coste se mantiene intacto. En cambio, cuando dicho nodo ya cuenta con dos enlaces dentro del árbol, significa que el siguiente enlace que agreguemos convertiría a ese vértice en switch. En ese momento se suma

inmediatamente a las aristas disponibles el peso adicional que se haya definido para un switch, de forma que el valor total del enlace resulte mayor. De esta manera, la comparación entre posibles enlaces deja de priorizar solo el coste de la conexión para empezar a penalizar automáticamente cualquier intento de darle al mismo nodo una tercera arista. Al llegar a su cuarta o quinta conexión, el nodo ya se considera switch y no recibe recargos adicionales, pues su coste extra solo se aplica una vez, justo cuando excede las dos conexiones.

Una vez que cada enlace candidato ha sido valorado, se selecciona aquel cuya suma de coste original más penalización, si se aplica, sea la más baja. Entonces ese enlace se incorpora al árbol, el nuevo nodo se marca como visitado, y se incrementa el contador de conexiones tanto del vértice de partida como del de llegada. Si justo antes de añadir ese enlace el vértice de partida tenía dos conexiones, en ese instante se acumula en un contador separado el importe total correspondiente a su penalización por switch. De esta forma, el algoritmo construye paso a paso un árbol que minimiza los sobre costes que surjan en el momento en que cada nodo supera las dos conexiones.

Cuando todos los vértices han sido agregados y el árbol está completo, se dispone de una topología en la que se ha evitado, en la medida de lo posible, sobrecargar un mismo nodo con tres o más conexiones. Cada vez que hubiera un intento de concentrar tráfico en un solo vértice, el coste incrementado de ese tercer enlace habría empujado al algoritmo a elegir otra alternativa. El resultado es una estructura de red que, aunque algunos enlaces puedan ser ligeramente más costosos que en la versión clásica de Prim, se logra reducir el número de switches, distribuyendo las conexiones entre más nodos y, por tanto, obteniendo una solución más equilibrada desde el punto de vista operativo y de instalación.

2.5. HEURÍSTICO N1

Este algoritmo se basa en repartir de forma equitativa el coste del switch de cada nodo con tres o más conexiones entre todas las aristas que lo conectan en la solución inicial. El objetivo de esta estrategia es generar una segunda evaluación de costes en la que aquellos nodos que en el árbol tradicional de Prim aparecieron con grado elevado vean sus costes de enlace incrementados de manera proporcional, de modo que, al volver a ejecutar Prim sobre esta matriz ajustada, se favorezca la selección de rutas que desplacen aristas a nodos con menos conexiones.

Pseudocódigo del Algoritmo 3 – HEURÍSTICO N1

Input:
- dir_in ← carpeta con archivos "*_RED.txt"
- output_excel ← ruta y nombre del Excel de salida

1. files ← list of "*_RED.txt" in dir_in; si está vacío imprime un aviso y retorna
2. summaries, edges ← empty lists
3. For each f in files do
   a. (n, E, W) ← READ_RED_FILE(f)
   b. C ← BUILD_COST_MATRIX(n, E)
   c. (T0, _) ← PRIM(C)
   d. S ← {u | degree of u in T0 ≥ 3}
   e. extra[u] ← W[u]/degree(u) for u in S
   f. C' ← C con cada arista en T0 incrementada por la suma de extras de sus extremos
   g. (T1, _) ← PRIM(C')
   h. (T, cost_arcs) ← REVERT_MST_TO_ORIGINAL(T1, C)
   i. cost_nodes ← ∑_{u∈{nodes of degree≥3 in T}} W[u]
   j. Añadir registros de sumarios y aristas de T a sumarios y aristas
4. Escribir en output_excel con hojas "Resumen" and "Aristas"

End of Pseudocódigo del Algoritmo 3

El procedimiento comienza ejecutando el algoritmo de Prim sin ninguna penalización de nodos, de modo que se obtiene un primer árbol de expansión mínima basado únicamente en los pesos originales de los enlaces. A partir de él se calcula, para cada nodo, cuántas conexiones ha adquirido; aquellos que alcanzan tres o más aristas se etiquetan como candidatos.

Después, se construye una nueva matriz de costes. Para ello, se recorre cada enlace que conecta a dos nodos cualquiera en la topología original. Si un extremo de ese enlace aparece en el árbol clásico con grado tres o superior, se suma a su coste original una porción del coste de switch de dicho nodo, pero no la totalidad. Concretamente, ese coste de switch se reparte uniformemente entre todas las aristas que originalmente conectan a ese nodo dentro del árbol clásico. De esta manera, cada enlace que parte de un nodo con tres o más conexiones ve su peso aumentado en la

parte proporcional de la penalización, mientras que los enlaces que conectan únicamente a nodos de grado cero, uno o dos mantienen su peso sin modificación.

Una vez completada esta redistribución, se vuelve a ejecutar el algoritmo de Prim, pero ahora utilizando la matriz de costes modificada. Debido a que las aristas que salen de los nodos con alto grado han sido encarecidas en proporción al número de conexiones que tenían, el algoritmo tenderá a evitar seleccionarlas si encuentra rutas alternativas ligeramente más costosas en la versión original, favoreciendo la incorporación de aristas que conecten nodos que inicialmente no eran conmutadores. Así, la nueva solución tenderá a equilibrar mejor la carga entre los vértices, reduciendo la cantidad de switches que superan tres conexiones en el árbol resultante.

Al concluir la segunda ejecución de Prim, se obtiene un árbol en el que algunas aristas han cambiado con respecto a la primera versión. Para calcular el coste final de la solución, primeramente se toma de nuevo cada vértice del árbol final y se determina su grado. Aquellos que continúen mostrando tres o más conexiones se consideran switches definitivos y se asume el coste completo. Paralelamente, se toma el peso original de cada arista sin los incrementos que se aplicaron durante la segunda ejecución del algoritmo de Prim. El coste global de la red viene dado, finalmente, por la suma de todos los enlaces que aparecen en el árbol final, más la suma de los costes de todos los nodos que actúan como switch en esa solución final.

El método persigue lograr una disminución sustancial en el número de switches con grado tres o superior, a costa de que algunos enlaces puedan ser un poco más costosos, pero consiguiendo un diseño de red más equilibrado y menos concentrado en puntos singulares de conmutación.

2.6. HEURÍSTICO N2

El propósito de este algoritmo es penalizar a los conmutadores distribuyendo el coste de switch entre todas las aristas del grafo original, independientemente de si formaron parte o no del árbol de Prim inicial. Con ello, se pretende obtener una valoración más homogénea del impacto que tiene un nodo sobre la topología completa, de modo que el algoritmo tienda a enfatizar rutas que utilicen nodos con menor grado global.

Pseudocódigo del Algoritmo 4 – HEURÍSTICO N2
Input:
• directorio con archivos "*_RED.txt"
• output_excel ruta de salida del Excel
1 archivos ← ficheros de directorio que acaban en "_RED.txt"
2 para cada archivo en archivos:
3 (n, aristas, pesos) ← READ_RED_FILE(ruta completa de archivo)
4 M ← BUILD_COST_MATRIX(n, aristas)
5 grados ← COMPUTE_GRAPH_DEGREES(aristas, n)
6 switches ← nodos con grado ≥ 3 en grados
7 extra ← { s: pesos[s−1] para s en switches }
8 M_mod ← M con suplemento extra en toda arista incidente a switches
9 (mst, cost_aristas) ← PRIM(M_mod)
10 (final_mst, cost_orig) ← REVERT_MST_TO_ORIGINAL(mst, M)
11 pen ← suma de pesos[s−1] para cada s en final_mst con grado ≥ 3
12 registrar resumen: [archivo, cost_orig, pen, cost_orig+pen, switches finales]
13 registrar aristas de final_mst
14 crear DataFrames de resumen y aristas y guardarlos en output_excel
• **READ_RED_FILE** lee n, lista de aristas y pesos de cada nodo.
• **BUILD_COST_MATRIX** construye la matriz de costes a partir de la lista de aristas.
• **COMPUTE_GRAPH_DEGREES** computa el grado de cada nodo en el grafo completo.
• **PRIM** obtiene el árbol de expansión mínima sobre la matriz modificada.
• **REVERT_MST_TO_ORIGINAL** restaura los costes originales de las aristas elegidas.
End of Pseudocódigo del Algoritmo 4

Para aplicar este método heurístico se parte de ejecutar Prim tradicional sobre la matriz de costes originales, de modo que se obtiene un árbol clásico sin penalizaciones en los nodos. A partir de ese primer resultado se calcula, para cada nodo, cuántas conexiones adquirió en el árbol final. Sin embargo, a diferencia del algoritmo Heurístico N1, este método no reparte el coste del switch únicamente entre las aristas que forman parte de ese árbol. En su lugar, distribuye el coste entre todas las aristas en las que participa dentro del grafo completo.

Para llevarlo a cabo, primero se determina el número total de aristas del grafo completo. Después, se detectan todos los nodos que en el árbol clásico alcanzaron un grado de tres o más; a cada uno de ellos se le asigna su peso de switch completo, sin repartirlo todavía. Luego, al construir la nueva matriz de costes, se recorre toda la lista

de enlaces original. Si alguno de los dos extremos aparece como nodos con grado alto en el árbol inicial, es decir, si en el árbol clásico tenían tres o más conexiones, se le suma al coste original de ese enlace la fracción correspondiente de todos los nodos que se hayan catalogado como switches. Esa fracción se obtiene dividiendo el coste del switch entre el total de aristas del grafo completo; de ese modo, cada enlace incidente a un switch recibe un pequeño sobre coste, idéntico en todos los casos, que refleja el peso del nodo como coste total dividido entre todas sus posibles líneas de comunicación. Como consecuencia, cuanto mayor sea el grado de un nodo dentro del árbol tradicional, más probable resulta que aparezca en la lista de switches.

Una vez completada esta redistribución de costes, se vuelve a ejecutar Prim, ahora sobre la matriz ajustada de manera que cualquier enlace que involucre a un nodo con al menos tres conexiones en la primera solución tenga un peso ligeramente superior. Dado que el sobre coste se suma de manera uniforme entre todas las aristas que conectan a un switch en el grafo completo, Prim tiende a elegir primero aquellas rutas que discurran por vértices que dejaron de ser considerados switches en la etapa previa o que, si eran switches, sus enlaces recibieron un valor menor comparado con otros enlaces penalizados de nodos más críticos.

Al terminar la ejecución de Prim sobre la matriz modificada, se obtiene un nuevo árbol en el que algunas aristas podrán haber sido reemplazadas por otras que, aunque su peso original fuera algo más elevado, resultan ser más eficientes al no involucrar switches. Para calcular el coste final de la solución, se procede de manera análoga al algoritmo Heurístico N1; se revisa cada vértice del árbol resultante y aquellos cuya conectividad nuevamente alcance tres o más conexiones son considerados switches definitivos. A esos nodos se les añade el coste completo del switch. Al mismo tiempo, se suman los pesos originales de todas las aristas del árbol sin tener en cuenta los recargos aplicados durante la ejecución, obteniendo el gasto real en cada enlace más los costes de instalar cada switch.

Este método logra una estructura más repartida ya que aquellos nodos que, en el árbol clásico, aparecieron con muchas conexiones, provocan recargos uniformes en todos sus enlaces potenciales, con lo cual Prim busca de forma natural rutas alternas que

usen nodos con menor grado global. En consecuencia, la nueva topología de conmutación favorece configuraciones más equilibradas sin depender únicamente de las conexiones específicas de la primera ejecución.

2.7. HEURÍSTICO N3

Este algoritmo introduce la aleatoriedad en la distribución del coste de switch entre las aristas incidentes a cada nodo de grado superior a dos, con el fin de explorar un mayor número de configuraciones y alejarse de óptimos locales. Mientras que en los métodos heurísticos N1 y N2 el recargo de switch se reparte de forma determinista, en este algoritmo se asigna, para cada nodo que alcanzó tres o más conexiones en la primera ejecución de Prim, una porción aleatoria de su coste total a cada una de sus aristas. De esta manera, cada iteración del algoritmo simula un reparto diferente del peso del switch, generando múltiples árboles candidatos entre los cuales se escogerá el de menor coste global.

Pseudocódigo del Algoritmo 5 – HEURÍSTICO N3
Input:
• directorio con archivos "*_RED.txt"
• output_excel ruta de salida del Excel
• iteration_values lista de iteraciones a evaluar
• max_workers número de procesos paralelos (opcional)
1 archivos ← ficheros en directorio con sufijo "_RED.txt"
2 para cada archivo en archivos:
3 for cada it ∈ iteration_values:
4 (n, aristas, pesos) ← READ_RED_FILE(archivo)
5 M ← BUILD_COST_MATRIX_INF(n, aristas)
6 switches ← nodos con grado≥3 en PRIM_SCIPY(M)
7 extra_partition aleatorio sobre aristas incidentes a switches
8 M_mod ← M + extras distribuidos (vectorizado)
9 (mst, _) ← PRIM_SCIPY(M_mod)
10 (final_mst, cost_aristas) ← REVERT_MST_TO_ORIGINAL(mst, M)
11 penalización ← suma de pesos de switches en final_mst
12 registrar resumen y aristas con campo "Iteraciones" = it
13 agrupar todos los resultados y escribirlos en output_excel
• **READ_RED_FILE:** lee número de nodos, aristas y pesos de nodo.
• **BUILD_COST_MATRIX_INF:** crea matriz de costes.
• **PRIM_SCIPY:** obtiene MST con SciPy.
End of Pseudocódigo del Algoritmo 5

El procedimiento comienza de la misma manera que los métodos anteriores: se ejecuta primero Prim usando únicamente los costes originales de los enlaces, obteniendo un árbol de expansión mínima sin penalizaciones en los nodos. A partir de dicho árbol se identifican todos los vértices que en esa solución tengan mínimo tres conexiones; estos vértices son considerados potenciales switches, y para cada uno conocemos el peso completo que se le ha asignado como penalización por ser switch.

Posteriormente, se realizan hasta mil iteraciones independientes en las que, en cada una de ellas, se genera una versión ligeramente distinta de la matriz de costes. Para lograrlo, se recorre cada uno de los nodos marcados como switch; suponiendo que un nodo concreto presenta k aristas en el grafo general independientemente de si esas aristas pertenecían o no al árbol inicial. En la iteración en curso, se extraen k valores aleatorios uniformes entre cero y uno, y se normalizan para que su suma sea exactamente igual a uno. A continuación, cada valor normalizado multiplica el peso total del switch de ese nodo, de modo que cada una de las k aristas reciba un sobre coste que, al sumarse, equivale al coste completo del switch. Como resultado, en cada iteración las aristas de un mismo switch pueden recibir pesos diferentes. Para los vértices que no superaron tres conexiones en el árbol inicial, no se aplica ninguna modificación y sus enlaces conservan siempre el peso original.

Una vez distribuidos los costes de forma aleatoria en una iteración determinada, se construye la matriz modificada, en la que cada enlace que conecta a un switch tiene su peso original aumentado por la porción correspondiente generada aleatoriamente. Con esta matriz recién creada, se ejecuta de nuevo Prim para obtener un árbol candidato.

Al concluir la ejecución de Prim en esa iteración, se obtiene un árbol en el que, nuevamente, se evalúa qué nodos han alcanzado grado tres o superior. En la mayoría de los casos coincidirán con los nodos ya identificados en la primera iteración, puesto que se busca modificar únicamente la evaluación de costes en los nodos con grados altos. Para calcular el coste total de la solución candidata se procede de la manera habitual: por un lado, se suman los pesos originales de todas las aristas que componen el árbol y, por otro, se suman los costes completos de cada nodo que en esta solución final aparezca con grado tres o más.

El paso siguiente consiste en comparar el coste total de este árbol contra el mejor árbol registrado hasta el momento en iteraciones anteriores. Si el nuevo árbol arroja un coste global inferior, entonces sustituye al mejor de referencia, si no, se descarta y se continúan las iteraciones.

2.8. MÉTODO GENÉTICO

A grandes rasgos, esta técnica emula un proceso de evolución ya que se genera una población inicial de soluciones candidatas, cada una codificada de manera que represente una posible asignación de costes en la topología. A continuación, se aplican operadores de cruce y mutación para producir descendientes que combinan características de sus "padres" y, a su vez, introducen pequeñas variaciones. Tras evaluar el desempeño de cada individuo, valorando su coste combinado de enlaces y switches, se seleccionan los más aptos para formar la siguiente generación. Este ciclo se repite durante un número de iteraciones predefinido, de manera que las soluciones tienden a mejorar progresivamente, aproximándose a configuraciones de coste cada vez menor.

Pseudocódigo del Algoritmo 6 – MÉTODO GENÉTICO
Input:
• directorio con archivos "_RED.txt"
• output_excel ruta de salida del Excel
• num_initial tamaño de la población inicial
• num_mixes número de cruces a realizar
• max_workers procesos paralelos (opcional)
1 archivos ← lista de "*_RED.txt" en directorio
2 para cada archivo en archivos en paralelo:
3 (mix_results, best) ← PROCESS_FILE_GENETIC(archivo, num_initial, num_mixes)
4 registrar mix_results en hoja con nombre de archivo
5 añadir best a lista de resúmenes
6 escribir todas las hojas y la hoja "Resumen" en output_excel
• PROCESS_FILE_GENETIC:
Ejecuta GENETIC_ALGORITHM_FILE sobre el fichero
Etiqueta cada registro de mezcla con el nombre del fichero
Construye un diccionario resumen con el mejor coste final, coste del MST y penalización
Devuelve (nombre_fichero, mix_results, resumen)
• GENETIC_ALGORITHM_FILE:
Lee el grafo y extrae nodos y pesos
Construye la matriz de costes y obtiene el MST clásico para identificar switches
Genera una población inicial de candidatos repartiendo aleatoriamente el coste de cada switch entre sus aristas
Realiza num_mixes cruces: escoge dos padres al azar, promedia sus "genomas" y evalúa el hijo (MST + penalización)
Si el hijo mejora a alguno de los padres, lo sustituye en la población
Devuelve la lista de resultados de cada mezcla y el mejor candidato final
End of Pseudocódigo del Algoritmo 6

Para implementar el método genético en nuestro contexto, se escogen las siguientes decisiones de diseño:

Codificación del genoma. Cada candidato de la población se corresponde con un vector de valores aleatorios asociados a cada nodo que, en la primera ejecución de Prim, apareció con al menos tres conexiones. En la práctica, ese vector indica, para cada switch, cómo se reparte su penalización total entre las aristas incidentes. De esta forma, un individuo contiene la información necesaria para reconstruir la versión modificada de la matriz de costes tal y como se haría en el algoritmo Heurístico N3, en la que cada posición del vector guarda un pequeño porcentaje que, multiplicado por el peso del switch, determina el recargo aplicado a cada arista de dicho nodo.

Inicialización de la población. Para crear la población inicial, se repite un proceso muy similar al Heurístico N3: cada individuo genera, de manera aleatoria, una distribución de recargos para cada switch identificado en el árbol clásico. A continuación, se ejecuta Prim y se obtiene un árbol candidato. Se repite esto hasta obtener un número predefinido de soluciones, constituyendo la población inicial.

Operador de cruce. En cada generación, se eligen aleatoriamente parejas de padres dentro de la población. Para formar cada hijo, se toma la media aritmética de ambos padres (componente a componente). Es decir, si un switch concreto tenía en el padre A un reparto del 20% y en el padre B un reparto del 50%, el hijo heredará un 35% para ese switch. De este modo, el nuevo individuo combina información de ambos padres y su distribución de recargos es una mezcla de las dos anteriores. Justo después de formar ese vector hijo, se construye la matriz de costes y se ejecuta nuevamente Prim para recalcular el árbol candidato y su coste total. La aptitud del hijo se compara luego con la de los padres: si el hijo resulta ser una muestra mejor, sustituye al padre de mayor coste; en caso contrario, se descarta y ambos padres permanecen en la población.

Operador de mutación. Para evitar que la población converja demasiado rápido hacia soluciones locales, se introduce una pequeña probabilidad de mutación sobre cada gen en algunos descendientes. Esto consiste en añadir o restar un valor aleatorio

pequeño, seguida de una normalización para que la suma de porcentajes vuelva a ser uno. Tras añadir la mutación, se vuelve a ejecutar Prim para conocer el coste real de la solución mutada.

Criterio de parada. El proceso evolutivo se repite durante un número fijo de iteraciones. Al finalizar, el individuo con menor aptitud (coste total más bajo) se considera la mejor solución hallada por la población, y se extraen de él el árbol resultante, los switches definitivos y el coste asociado.

3. RESULTADOS Y DISCUSIÓN

3.1. DATOS EXPERIMENTALES

Para evaluar de manera rigurosa el comportamiento de los algoritmos desarrollados en este proyecto, se emplearon como base las instancias propuestas por Carrabs et al. [64], del que se basan los dos artículos principales como referencia a este proyecto [43] [63]. Estas instancias, diseñadas específicamente para el problema de Árbol Generador con Mínimo Número de Vértices de Ramificación (MBV), han sido adoptadas en numerosos trabajos posteriores como estándar de referencia en pruebas computacionales. A lo largo de esta sección se describirá con todo detalle cómo se generan las instancias propuestas por Carrabs et al. [64] y qué características estructurales presentan.

Las instancias contemplan dos categorías principales según el número de vértices n:

- Instancias medianas. Abarcan valores de n en el conjunto (20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 250, 300, 350, 400, 450, 500). Para cada valor de n en este rango, se generaron 25 grafos independientes, de modo que al final existieran 25 variantes de cada tamaño. Esto da como resultado un total de 16 tamaños distintos y 25 réplicas, 400 instancias “medianas”.
- Instancias grandes. A partir de n=600 se pasa a la categoría de instancias que, por su tamaño, resultan más desafiantes para las técnicas exactas (600, 700, 800, 900, 1000). Se volvieron a generar 25 grafos para cada uno, sumando otras 125 instancias, que en conjunto con las 400 anteriores conforman un total de 525 grafos. Estas instancias grandes se definieron con el fin de analizar la escalabilidad de los algoritmos: las técnicas exactas comenzaron a tener dificultades para resolverlas en un límite de tiempo de una hora, mientras que las heurísticas continuaban proporcionando soluciones de buena calidad, permitiendo así contrastar eficiencia y calidad en escenarios de gran envergadura.

Respecto al proceso de generación aleatoria y conectividad, para cada par (n,m) (número de nodos y número de aristas), realizaron la siguiente secuencia:

- Comenzando con el conjunto de vértices {1, 2, ..., n}, se seleccionaron m pares (u,v) sin orden, de forma equiprobable sobre todas las combinaciones posibles, evitando duplicados.
- Cada vez que se escogía un par (u,v), se verificaba que u distinto de v y que la arista (u,v) no existiera previamente. Este proceso se repetía hasta completar m aristas.

Tras elegir esas m aristas, se comprobaba si el grafo era conexo, es decir, si existía al menos un camino entre cualquier par de vértices. En caso de quedar dividido en más de un componente, se descartaba la réplica actual y se generaba un nuevo conjunto aleatorio de aristas hasta obtener un grafo conectado.

En cuanto a la asignación de costos a aristas y nodos, las aristas no disponían de peso asociado; el objetivo era simplemente minimizar el número de vértices de ramificación. No obstante, para su heurístico inicial, introdujeron un peso ficticio de valor 1.0 a todas las aristas, de modo que el Prim modificado pudiera emplearse para construir árboles de expansión mínima en coste uniforme.

El peso de cada nodo no se asignó en esta generación aleatoria, sino que cada vértice se considera uniformemente penalizado cuando alcanza a ramificar.

Para integrar las instancias en nuestro entorno de experimentación, se siguieron estos pasos:

Respecto a la lectura de datos, cada archivo sigue un estándar, la primera línea contiene dos números enteros y un “0”. El formato es: <n> <m> 0, donde <n> y <m> son, respectivamente, el número de vértices y aristas.

Las siguientes m líneas contienen números de la forma u v w, donde u y v ($1 \leq u,v \leq n$) definen cada arista (u,v) y w es un peso generado por un programa que asigna valores entre 10 y 100. En los algoritmos desarrollados, se leen esas líneas para construir la estructura de lista de adyacencia y la matriz de costes. Se crea una matriz de adyacencia de tamaño n x n, inicialmente con ceros. Cada arista (u,v) registrada con peso w se traduce en asignar en la posición [u-1][v-1] y [v-1][u-1] el valor w.

Cada grafo se almacena en dos formatos internos:

- Lista de aristas: colección de tuplas (u,v,w), para alimentar directamente los heurísticos (por ejemplo, la heurística N3 necesita un array de aristas para distribuir penalizaciones).
- Matriz de costes: matriz bidimensional de tamaño n x n con coste infinito para pares no adyacentes y con coste w para pares adyacentes.

3.2. HERRAMIENTAS DE PROGRAMACIÓN Y ANÁLISIS

Para garantizar flexibilidad, legibilidad y escalabilidad, se optó por utilizar el lenguaje Python en su versión 3.13.1, junto con un conjunto de librerías especializadas que facilitan la manipulación de estructuras de datos, la realización de cálculos matriciales y vectoriales, la generación de árboles de expansión mínima, la ejecución de procesos en paralelo y la visualización de resultados.

Python 3 ha sido la columna vertebral de toda la implementación. La elección de Python se fundamenta en varias razones: cuenta con una sintaxis clara y concisa que favorece el desarrollo rápido de prototipos, ofrece un amplio ecosistema de librerías científicas y de análisis de datos, y dispone de módulos integrados para concurrencia y ejecución paralela.

Para las operaciones matriciales y vectoriales, se ha utilizado la librería NumPy (Numerical Python). NumPy proporciona la estructura ndarray, un arreglo multidimensional que puede almacenar datos numéricos de diversos y sobre el que se implementan operaciones elementales y lineales de forma optimizada en C. En este caso, la representación de las matrices de costes de la red se basa en ndarray de NumPy. En ella las matrices de adyacencia se almacenan como arreglos bidimensionales, donde cada elemento (i, j) guarda el coste o la latencia del enlace entre los nodos i y j. Gracias a la capacidad de NumPy para vectorizar operaciones, se pudo modificar rápidamente los valores de los enlaces conforme a los criterios de los métodos heurísticos, sin recurrir a bucles explícitos en Python, lo cual habría supuesto una pérdida considerable de rendimiento. Además, NumPy ofrece funciones de creación de arreglos que se utilizaron para inicializar matrices de costes y vectores de peso de nodos de forma directa y con un mínimo de código.

Sobre NumPy se construye la librería SciPy, que agrupa numerosas funciones de análisis científico, desde métodos de optimización y ajuste de curvas hasta procesamiento de señales y álgebra lineal avanzada. En particular, se aprovechó el submódulo scipy.sparse.csgraph para calcular el árbol de expansión mínima (Minimum Spanning Tree, MST) de forma eficiente sobre grafos que podían incluir miles

de nodos. La función minimum_spanning_tree de SciPy opera sobre matrices dispersas, reduciendo significativamente el uso de memoria cuando las redes presentan una gran cantidad de posibles enlaces no existentes.

Para el manejo de datos tabulares y la generación de informes consolidados, se recurrió a Pandas, una librería que pone a disposición estructuras de datos como DataFrame y Series. Gracias a esta librería, los resultados de cada iteración de los algoritmos pudieron almacenarse directamente en el módulo DataFrame y exportarse a archivos con formato Excel. Se hicieron usos frecuentes de métodos como to_excel(), to_csv() o merge() para consolidar información de múltiples experimentos en un único archivo con varias hojas, clasificado por tipo de algoritmo. Esta capacidad de Pandas para cargar, filtrar y exportar datos permitió ordenar la fase de análisis de resultados y preparar hojas de cálculo incluidas en el anexo de este trabajo.

Dado que algunos componentes del proyecto requerían ejecutar numerosos experimentos independientes, fue fundamental aprovechar la capacidad de Python para ejecutar trabajos en paralelo. Para ello, se utilizó el módulo estándar concurrent.futures, en especial la clase ProcessPoolExecutor, que permite ejecutar tareas en diferentes procesos del sistema operativo, aprovechando múltiples núcleos de CPU. Cada experimento se encapsuló en una función que luego se envió a un conjunto de procesos de forma asíncrona. Gracias a esta paralelización, se redujo drásticamente el tiempo total de cálculo, permitiendo probar redes de varias centenas de nodos en un tiempo razonable. La combinación de concurrent.futures con la librería tqdm, la cual añade barras de progreso a bucles y a la espera de futuros, de modo que se podía monitorizar visualmente, en tiempo real, cuántas iteraciones de un experimento permanecían pendientes y cuántos habían finalizado. Esto facilitó el seguimiento durante largas ejecuciones.

El procesamiento de grandes volúmenes de datos precisa controlar el uso de memoria. Para ello, se complementó el uso de NumPy con la librería SciPy.sparse, cuyo módulo csr_matrix (Compressed Sparse Row) y csc_matrix (Compressed Sparse Column) permitieron conservar la matriz de costes en un formato compacto cuando era necesario. En particular, cuando se deseaba aplicar Prim sobre redes muy dispersas,

se transformaba la matriz de NumPy en un csr_matrix para alimentar directamente al método minimum_spanning_tree de SciPy.

Finalmente, para tareas de lectura y escritura de archivos, se dispuso de operaciones de lectura línea por línea, cadenas y métodos de conversión de tipos (split(), int(), float()). Sin embargo, para generar reportes consolidados en un único archivo Excel que incluyese varias hojas se utilizó ExcelWriter junto con el motor openpyxl. De esta manera, fue posible crear un solo fichero .xlsx que al abrirlo mostrara pestañas dedicadas a cada experimento, facilitando enormemente la revisión de resultados.

3.3. PRESENTACIÓN DE LOS RESULTADOS Y ANÁLISIS COMPARATIVO

En este apartado, se expone de manera clara y detallada todos los datos obtenidos, donde se muestra la media obtenida en cada conjunto de nodos, partiendo de datos iniciales obtenidos con el algoritmo de Prim clásico para disponer de una buena base de comparación con el resto de los métodos desarrollados.

NODOS	CLÁSICO	DINÁMICO	HEU - N1	HEU - N2	HEU - N3	GENÉTICO
20	334,24	187,96	373,76	238,28	110,04	108,36
40	754,76	523,96	785,16	528,16	308,88	306,2
60	1191,28	909,84	1250,64	939,76	593,08	579,8
80	1590,44	1238,12	1608,88	1231,48	876	848,76
100	2037,72	1701,24	2098,8	1694,72	1227,12	1207,48
120	2434,88	2051,48	2507,04	2032,64	1578,6	1547,92
140	2876,72	2433,2	2920,48	2386,88	1862,2	1823,04
160	3350,72	2866	3335,04	2784,28	2285,2	2241,56
180	3757,56	3245,44	3782,16	3115,8	2586,28	2520,84
200	4195,68	3593,08	4176,84	3500,68	2914,4	2855,44
250	5280,72	4639	5295,56	4550,88	3893,52	3812,2
300	6442,72	5697,16	6475,76	5615,36	4846,72	4749,72
350	7566,64	6755,28	7437,64	6553,36	5806	5704,96
400	8718,88	7902,16	8551,04	7705,44	6979,12	6857,84
450	9792,92	8864,48	9597,96	8687,64	7875,24	7734,48
500	10879,36	9906,24	10654,48	9679,64	8909,48	8750,92
600	13863,32	13037,84	14366,88	13210,36	11935,48	11824,8
700	16116,12	15223,32	16600,12	15316,04	14015,76	13868,4
800	18481,36	17569,6	18912,4	17626,2	16132,72	15979,4
900	20877,8	19865,6	21448,04	19988,16	18461,48	18271,96
1000	23269,64	22255,68	23859,24	22382,4	20681,56	20511,48
MEDIA	7800,64	7165,08	7906,57	7131,82	6375,18	6290,74

Tabla 1. Comparativa de la media de resultados obtenidos según la cantidad de nodos de la red.

Al observar la evolución de los valores de coste medio en función del tamaño de la red recogido en la Tabla 1, se observa que la realización de un método dinámico al algoritmo de Prim ya aporta una mejora notable sobre la versión clásica. En redes de tamaño reducido, la reducción de coste alcanza magnitudes muy destacables, mientras que en grafos de mil nodos el descenso, aunque menor en porcentaje, sigue siendo apreciable. Este comportamiento sugiere que posponer la penalización hasta la tercera conexión obliga al algoritmo a explorar alternativas durante las primeras etapas de construcción del árbol, algo especialmente beneficioso cuando el número de rutas posibles es limitado; en redes muy grandes, la abundancia de caminos de bajo coste hace que ese recargo puntual tenga un impacto más moderado.

El método Heurístico N1, de reparto equitativo y global ofrecen resultados matizados. Cuando el peso de cada switch se distribuye únicamente entre sus aristas incidentes, la mejora nunca supera al recargo dinámico directo, en ninguna instancia. En cambio, en el Heurístico N2, al repartir el coste del switch entre todas las aristas del grafo, se logra una penalización más homogénea que estabiliza las prioridades de Prim y genera árboles con un número de switches muy similar o ligeramente inferior al obtenido por la versión dinámica en instancias de cualquier tamaño. Un reparto demasiado localizado puede sesgar las decisiones de incorporar o no un determinado enlace, mientras que una distribución amplia amortigua el efecto en redes densas.

El algoritmo Heurístico N3, que incorpora un reparto aleatorio de penalizaciones entre las conexiones de cada switch, consigue el coste medio más bajo fuera del método genético. Al variar en cada iteración la forma en que cada nodo de alto grado reparte su peso entre sus enlaces, se explora un abanico mucho más amplio de configuraciones posibles. Este proceso de búsqueda estocástica capta soluciones que ni los métodos deterministas ni los ajustes graduales pueden alcanzar, especialmente en instancias de tamaño medio donde la complejidad de rutas posibles crece de forma acelerada. Finalmente, el método genético perfecciona aún más este enfoque al combinar genomas prometedores y seleccionar repetidamente las mejores soluciones. Aunque la ganancia incremental sobre la heurística aleatoria es modesta, se produce de manera constante en todas las redes evaluadas, confirmando que la evolución de

poblaciones de recompensas conduce a árboles con el número mínimo de switches detectado en cada escenario.

NODOS	CLASICO	DINAMICO	HEU - N1	HEU - N2	HEU - N3	GENÉTICO
20	269,84	106,8	274,52	139,16	33,2	31,52
40	607,76	342,88	568,8	320,56	131,6	128,92
60	952,12	627,16	907,8	622,4	307,32	294,04
80	1259,8	854,2	1141,72	805,72	488	460,76
100	1592,12	1196,28	1505,24	1144,52	714,24	694,6
120	1910,44	1451,96	1805,92	1390,32	974,16	943,48
140	2260,44	1736,4	2110,08	1644,12	1155,24	1116,08
160	2650,08	2080	2413,36	1936,28	1488,6	1444,96
180	2940,36	2329,6	2718,32	2137,44	1661,97	1596,53
200	3280,16	2573,88	3011,8	2410,96	1884,84	1825,88
250	4111,64	3345,12	3839,2	3188,16	2597,4	2516,08
300	5041,44	4157,2	4748,8	3997,72	3299,24	3202,24
350	5915,2	4952,6	5429,96	4669,32	3996,08	3895,04
400	6797,64	5815,08	6261	5529,2	4895,16	4773,88
450	7639,28	6528,76	7033,2	6247,08	5533,6	5392,84
500	8438,64	7315,72	7788,08	6948,08	6274,76	6116,2
600	10890,68	9886,4	10926,32	9946,84	8773,04	8662,36
700	12621,32	11531,68	12604	11507,68	10322,12	10174,76
800	14447,64	13316,72	14331,4	13242,68	11869,84	11716,52
900	16334,84	15106,76	16340,72	15088,04	13691,28	13501,76
1000	18221,96	16965,56	18211,68	16952,8	15381,68	15211,6
MEDIA	6103,97	5343,85	5903,42	5231,86	4546,35	4461,91

Tabla 2. Coste de conmutadores medio según la cantidad de nodos de la red.

NODOS	CLASICO	DINAMICO	HEU - N1	HEU - N2	HEU - N3	GENÉTICO
20	64,4	81,16	99,24	99,12	76,88	73,52
40	147	181,08	216,36	207,6	177,24	171,88
60	239,16	282,68	342,84	317,36	285,76	259,2
80	330,64	383,92	467,16	425,76	388,71	334,23
100	445,6	504,96	593,56	550,2	512,88	473,6
120	524,44	599,52	701,12	642,32	604,44	543,08
140	616,28	696,8	810,4	742,76	706,96	628,64
160	700,64	786	921,68	848	793,72	706,44
180	817,2	915,64	1063,84	978,36	925,28	794,4
200	915,52	1019,2	1165,04	1089,72	1029,56	911,64
250	1169,08	1293,88	1456,36	1362,72	1296,12	1133,48
300	1401,28	1539,96	1726,96	1617,64	1547,48	1353,48
350	1651,44	1802,68	2007,68	1884,04	1809,92	1607,84
400	1921,24	2087,08	2290,04	2176,24	2083,96	1841,4

450	2153,64	2335,72	2564,76	2440,56	2341,64	2060,12
500	2440,72	2630,52	2866,4	2731,56	2634,72	2317,6
600	2972,64	3151,44	3440,56	3263,52	3162,44	2941,08
700	3494,8	3691,64	3996,12	3808,36	3693,64	3398,92
800	4033,72	4252,88	4581	4383,52	4262,88	3956,24
900	4542,96	4758,84	5107,32	4900,12	4770,2	4391,16
1000	5047,68	5290,12	5647,56	5429,6	5836,04	5495,88
MEDIA	1693,60	1819,26	1998,42	1895,24	1850,65	1681,92

Tabla 3. Coste de conexiones medio según la cantidad de nodos de la red.

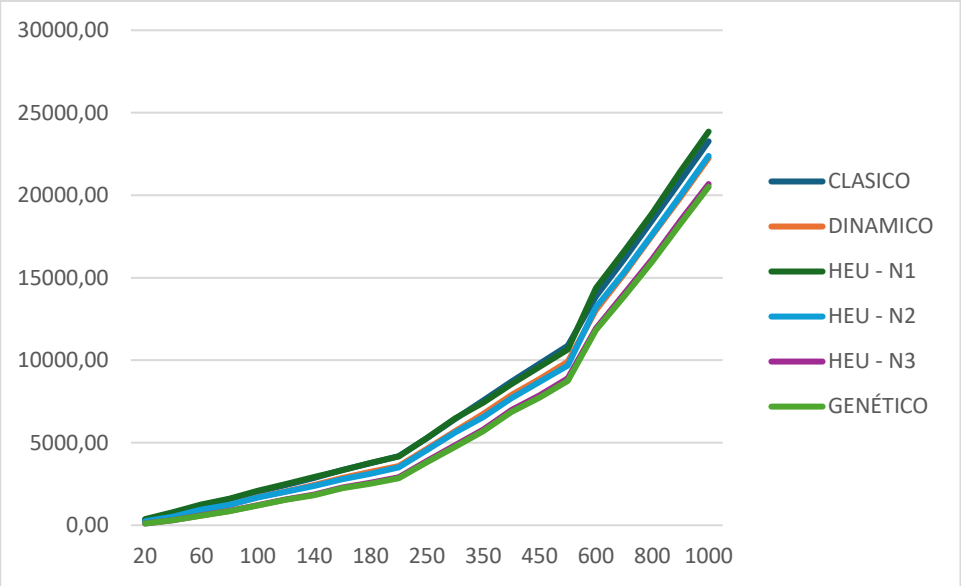
En la Tabla 2 se aprecia que Prim clásico se ve rápidamente superado por todos los algoritmos desarrollados. Mientras en redes de 20 nodos dispone de una media de 269,84 en coste de switches, Prim dinámico lo consigue reducir hasta 106,8 y los heurísticos bajo reparto aleatorio N3 y el genético llegan incluso a 33,2 y 31,52 respectivamente. A medida que la red crece, la diferencia se atenúa, pero la jerarquía de rendimiento permanece. El método genético ocupa siempre el primer lugar, seguido muy de cerca por N3, y con el algoritmo Dinámico y el N2 a una distancia moderadamente mayor.

Sin embargo, la mera reducción de nodos de alta conmutación no basta si ello implicara un alza sustancial del coste en enlaces. La Tabla 3 revela que Prim clásico, mantiene costes de conexión muy bajos en cualquier configuración de red. Los algoritmos heurísticos N2 y N3 equilibran mejor ambos factores, moderando el coste de enlaces. N1, a pesar de no sobresalir en conmutadores, tampoco logra mejorar los costes de conexión, siendo el que peores resultados medios arroja.

Los enfoques dinámicos relucen más en términos de costes de aristas respecto a los métodos heurísticos. Por ejemplo, en redes con 100 nodos, su coste de conexiones es de 504,96 frente a los 445,6 de Prim clásico, y al mismo tiempo reduce notablemente los costes por switches, pudiendo trasladar esta comparación con cualquier configuración.

El método genético, por su parte, entrega simultáneamente los menores valores medios en ambas métricas prácticamente siempre. En redes con 700 nodos, obtiene 10174,76 de coste en switches con 3398,92 de coste de conexiones, confirmando su

capacidad para explorar el espacio de soluciones y encontrar árboles de expansión mínima que optimizan a la vez la topología y la eficiencia del tránsito.

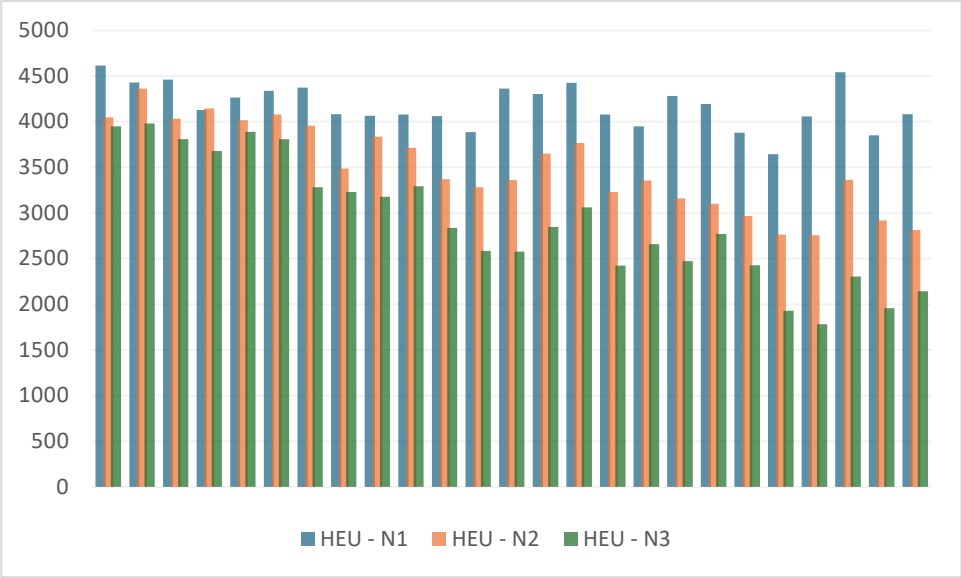


Gráfica 1. Evolución del coste total en cada algoritmo.

CLÁSICO	DINÁMICO	HEU - N1	HEU - N2	HEU - N3	GENÉTICO
REDES MEDIANAS (20 - 500)					
71205,24	62514,64	70851,24	61245	52651,88	51649,52
	12,21%	0,50%	13,99%	26,06%	27,46%
REDES GRANDES (600 - 1000)					
92608,24	87952,04	95186,68	88523,16	81227	80456,04
	5,03%	-2,78%	4,41%	12,29%	13,12%
1000 NODOS					
23269,64	22255,68	23859,24	22382,4	20681,56	20511,48
	4,36%	-2,53%	3,81%	11,12%	11,85%

Tabla 4. Comparativa del porcentaje de mejoría o empeoramiento de resultados de switches y aristas obtenido respecto al algoritmo de Prim clásico.

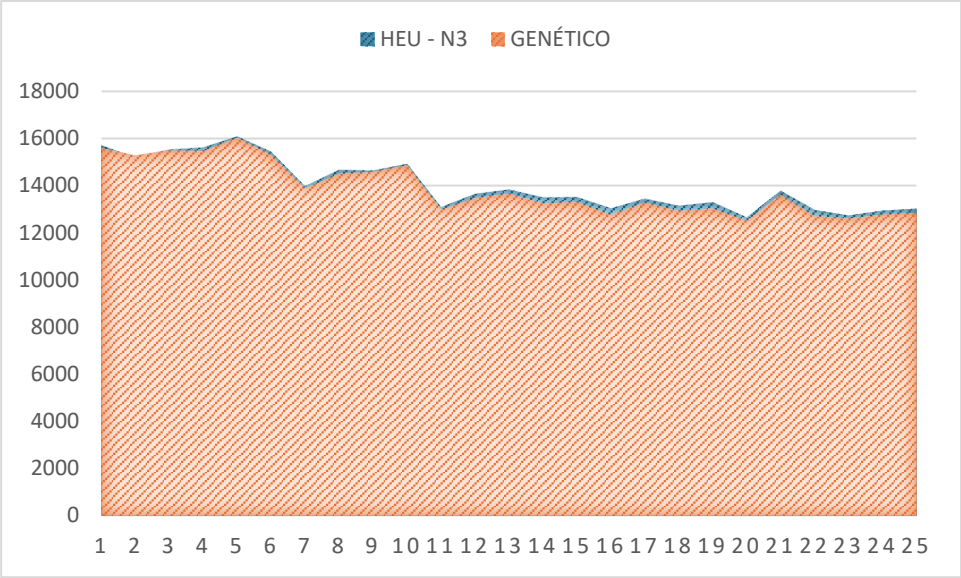
Observando los datos recogidos en la Tabla 4, podemos ver que el algoritmo de Prim dinámico obtuvo, en redes de tamaño mediano, una reducción de switches con valores próximos al 12% respecto a la versión estática de Prim. Este beneficio se traduce en árboles de expansión donde la mayoría de los nodos alcanza, como máximo, dos conexiones y solo muy pocos alcanzan grado tres o superior.



Gráfica 3. Evolución del coste en métodos heurísticos en instancias de 200 nodos.

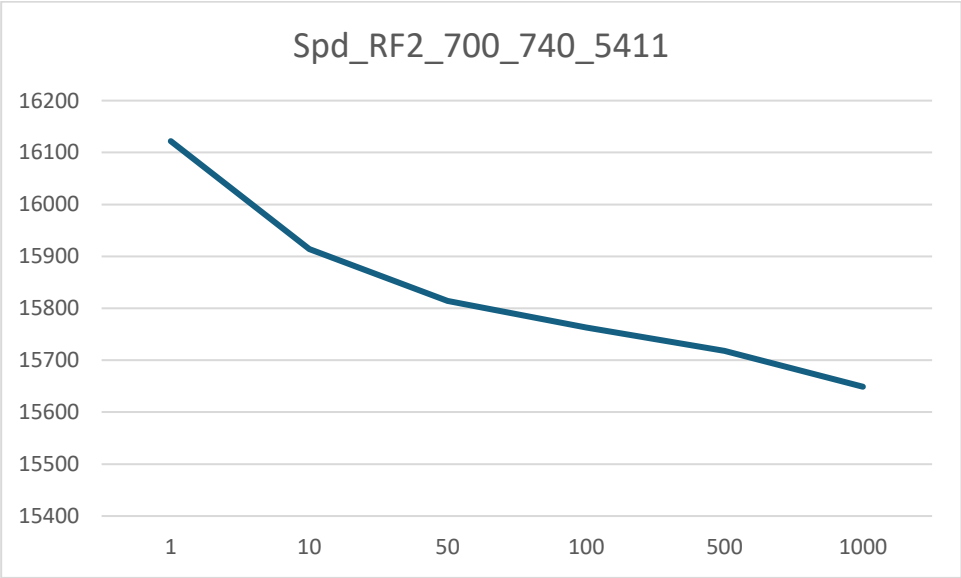
El algoritmo Heurístico N1 consiguió resultados similares al de Prim clásico en redes medianas, sin apenas mejoría. Sin embargo, en redes grandes obtuvimos los peores resultados, incluso peor que los proporcionados por el algoritmo clásico de Prim. Esto puede deberse a que el modelo se basa en repartir de forma equitativa el coste del switch de cada nodo con tres o más conexiones entre todas las aristas que lo conectan en la solución inicial, empeora la mejor solución con tal de minimizar la cantidad de conmutadores, a costa de empeorar el coste final.

El algoritmo Heurístico N2 implementa una penalización uniforme para todas las aristas conectadas a cada switch detectado en la primera iteración de Prim. Allí donde el método Heurístico N1 solo encarece las aristas que ya formaban parte del árbol previo, el Heurístico N2 aporta el coste de todo enlace sobre cualquier nodo marcado como switch. Como consecuencia, se halló rutas alternativas que muchas veces implicaban enlaces originales de coste algo mayor, pero obtuvieron reducciones de un 14% menos de switches en redes medianas.



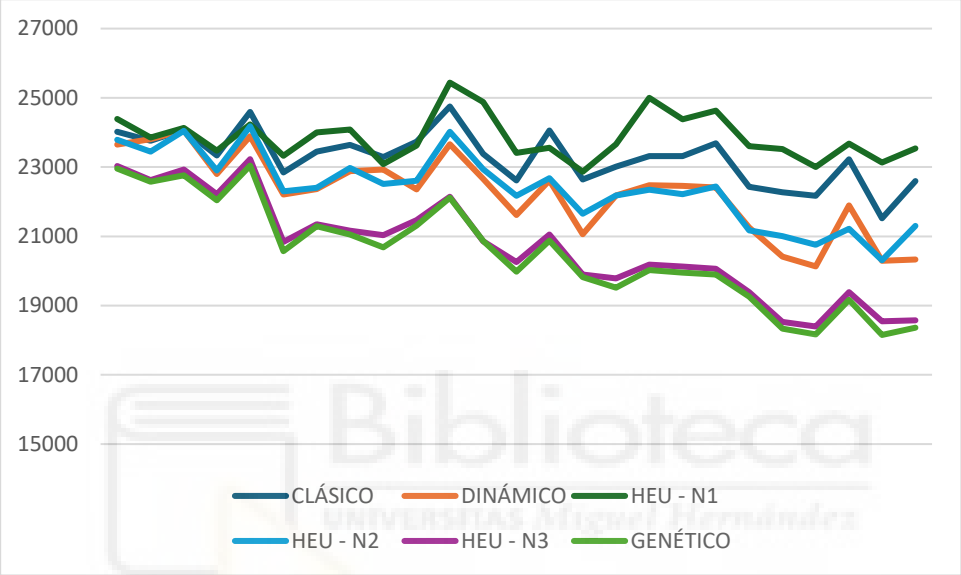
Gráfica 4. Comparación de resultados obtenidos entre el algoritmo HEU – N3 y el genético en instancias de 700 nodos.

El método Heurístico N3, que introduce aleatoriedad al repartir el coste de cada switch entre sus aristas incidentes, consiguió el mejor equilibrio entre coste de enlaces y reducción de switches del resto de algoritmos heurísticos. En particular, al repetir múltiples iteraciones, entre 20 y 500 nodos, y tomar la mejor solución global, alcanzó reducciones superiores al 26% de. Por otro lado, en una red de 1000 nodos con densidad de conexiones media, el número de switches se redujo en un 11% en la mejor ejecución del método en cada instancia. Las curvas de evolución del coste se aprecia la diferencia notable entre métodos heurísticos.



Gráfica 2. HEU – N3. Evolución del coste en función del número de iteraciones en una red de 700 nodos.

Finalmente, el método genético combinó las mejores características del método Heurístico N3 con un proceso evolutivo que produjo soluciones aún más afinadas. A lo largo de diversas mezclas, el genético fue capaz de encontrar árboles con un 13% menos de switches en redes de 600–1000 nodos, superando ligeramente al Heurístico N3.



Gráfica 5. Comparación de resultados obtenidos en redes de 1000 nodos

En las tablas detalladas del anexo se recogen todos los valores numéricos específicos de cada algoritmo para cada topología de tamaño variable se muestran el coste total que resulta de la suma del MST y de la aportación de cada switch en el árbol.

NODOS	CLASICO	DINAMICO	HEU – N1	HEU – N2	HEU – N3	GENETICO
20	14,39	47,95	119,88	122,84	409,46	431,05
40	28,77	95,91	239,77	245,68	818,92	862,24
60	43,16	143,86	359,65	368,51	1228,38	1293,17
80	57,54	191,81	479,53	491,35	1637,84	1724,09
100	71,93	239,77	599,42	614,19	2047,30	2155,93
120	86,32	287,72	719,30	737,03	2456,76	2586,41
140	100,70	335,67	839,18	859,86	2866,22	3017,62
160	115,09	383,63	959,07	982,70	3275,68	3448,71
180	129,47	431,58	1078,95	1105,54	3685,14	3879,43
200	143,86	479,53	1198,83	1228,38	4094,60	4310,66
250	179,82	599,42	1498,54	1535,47	5118,24	5387,94

300	215,79	719,30	1798,25	1842,57	6141,89	6465,15
350	251,75	839,18	2097,96	2149,66	7165,54	7542,68
400	287,72	959,07	2397,66	2456,76	8189,19	8620,20
450	323,68	1078,95	2697,37	2763,85	9212,84	9697,73
500	359,65	1198,83	2997,08	3070,95	10236,49	10775,23
600	431,58	1438,60	3596,50	3685,14	12283,79	12930,36
700	503,51	1678,36	4195,91	4299,32	14331,08	15085,31
800	575,44	1918,13	4795,33	4913,51	16378,38	17240,40
900	647,37	2157,90	5394,74	5527,70	18425,68	19395,88
1000	719,30	2397,66	5994,16	6141,89	20472,98	22049,61
MEDIA	251,754286	839,182381	2097,95619	2149,6619	7165,54286	7566,65714

Tabla 5. Coste de computación total en segundos según la cantidad de nodos de la red.

Al observar los tiempos de ejecución indicados en la Tabla 5 de los distintos métodos sobre redes de tamaños crecientes, se aprecia de manera clara cómo los algoritmos clásicos y dinámicos mantienen un comportamiento notablemente eficiente incluso al escalar hasta 1000 nodos. El algoritmo clásico incrementa su tiempo desde apenas catorce segundos en resolver 25 redes de 20 nodos hasta los doce minutos en 1000 nodos, mientras que la versión dinámica consigue tardar 48 segundos y acabando en poco más de 40 minutos en analizar todas las redes con 1000 nodos.

Por su parte, las heurísticas N1 y N2 siguen un patrón de crecimiento muy similar, con tiempos ligeramente superiores en N2. Ambas comienzan alrededor de 2 minutos para 20 nodos, y terminan en torno a una hora y 40 minutos para 1000 nodos.

La heurística N3, en cambio, muestra un crecimiento mucho más acusado. El tiempo de N3 arranca en 6 minutos para la red más pequeña y supera las cinco horas en la de 1000 nodos, proporcionando un tiempo de alrededor de 13 minutos para cada instancia de esta cantidad de nodos. Esto se debe a la generación aleatoria y normalización de particiones de coste en cada iteración, lo que multiplica el trabajo proporcionalmente al número de aristas incidentes. A pesar de esto, N3 ofrecía en la Tabla 1 reducciones de coste cercanas al 16% frente al clásico y equivalentes al genético, lo que revela una relación inversa entre calidad de la solución y tiempo de ejecución.

Finalmente, el método genético se sitúa como el más costoso en tiempo, pasando de 7 minutos a más de 6 horas al aumentar de 20 a 1000 nodos. No obstante, este esfuerzo

computacional se refleja en los mejores resultados de coste general, superando a todas las heurísticas en prácticamente todas las instancias. En consecuencia, para aplicaciones donde el tiempo no sea crítico o se disponga de potentes recursos de cálculo en paralelo, el método genético representa la alternativa más efectiva en términos de optimización global.

4. CONCLUSIONES

4.1. CONCLUSIONES GENERALES

Mientras que la versión original de Prim solo optimiza la suma de los pesos de las aristas, este trabajo demuestra que dicha estrategia adolece de un sesgo hacia soluciones que, pese a garantizar el mínimo coste de enlace, pueden concentrar el tráfico en un reducido conjunto de nodos, multiplicando la cantidad de switches instalados y, con ello, elevando la complejidad operativa.

Al introducir modificaciones simples pero efectivas en Prim, desde la aplicación de un recargo estático cada vez que un nodo superaba dos aristas hasta penalizaciones graduales o distribuidas proporcionalmente, se comprobó que es posible distribuir la carga de conmutación de manera mucho más homogénea, evitando la aparición prematura de nodos altamente ramificados. En particular, Prim dinámico demostró que basta con aplicar el recargo únicamente al momento de consolidación de un switch para obtener rutas alternativas sin necesidad de modificar drásticamente la lógica original del algoritmo. Este ajuste puntual produce una reducción significativa en el número total de switches: en grafos con densidades moderadas, se observaron disminuciones de hasta un 12 % con respecto a Prim clásico.

Con las adaptaciones heurísticas N1 y N2 se consiguió ahondar más en la mejora de la ramificación. La heurística N1, que reparte de forma equitativa el coste de cada switch entre todas sus aristas incidentes, consigue hacer que Prim perciba el impacto de un switch de un modo repartido e incremental. Gracias a ello, el algoritmo evita de forma más pausada y consciente el sobreuso de vértices anteriormente identificados como problemáticos.

La heurística N2 fue el paso siguiente en la evolución de estas ideas, ya que, en lugar de centralizar el reparto únicamente en las aristas incidentes a cada nodo, distribuye el coste completo del switch proporcionalmente al total de aristas presentes en el grafo. Esta ampliación del concepto de reparto busca potenciar la importancia relativa del switch en el conjunto de la red, de manera que cuantas más conexiones existiesen globalmente, menos gravoso resultara cada enlace en particular. Como resultado,

consiguió reducir el número total de switches hasta en un 1.5% frente a Prim dinámico, manteniendo un tiempo de cómputo comparable al de N1.

El algoritmo heurístico N3, por su parte, introdujo un componente clave de aleatoriedad. Al repartir de modo proporcional y aleatorio el peso del switch entre las aristas incidentes en cada iteración, el algoritmo explora un amplio espectro de alternativas topológicas: cada ejecución genera una matriz de costes distinta y, por tanto, un árbol de expansión mínima con posibles configuraciones de switches dispares. A lo largo de varias repeticiones, se selecciona la solución que arroje el menor coste total.

Por último, el método genético supone el culmen de este proceso de refinamiento. Combinando sistemáticamente genomas que representan diferentes formas de repartir el coste de cada switch y aplicando esquemas de cruce y selección, el método evolutivo logra converger a soluciones que reducen aún más la ramificación. En las pruebas, el enfoque genético fue capaz de igualar o mejorar ligeramente los mejores valores de N3, alcanzando reducciones de hasta un 8% respecto a Prim dinámico en grafos densos de 600–1000 nodos. Aunque su tiempo de ejecución es mayor, la ganancia en el número de switches para instancias de gran tamaño justifica la inversión computacional.

En definitiva, el conjunto de algoritmos presentados demuestra de manera inequívoca que la minimización del número de switches es una dimensión complementaria a la minimización de pesos de enlaces. Al integrar el coste de nodos de forma gradual, repartida o evolutiva, se generan topologías de red con menor ramificación sin perder conectividad. Estos avances constituyen un aporte significativo en el ámbito de la ingeniería de redes de proveedores de Internet, posibilitando infraestructuras más sencillas de gestionar, con menor latencia acumulada y menor complejidad operacional.

4.2. TRABAJOS FUTUROS Y PROPUESTAS DE MEJORA

Como línea de investigación futura, sería interesante abordar la extensión de los algoritmos a escenarios en los que la topología de la red no permanece estática, sino que evoluciona con el tiempo. En ambientes reales de proveedores de Internet, frecuentemente se añaden o retiran nodos y enlaces en función de fluctuaciones de tráfico, tareas de mantenimiento o migración de servicios a nuevos centros de datos. Para modelar este dinamismo, se requerirían mecanismos capaces de actualizar el árbol de expansión mínima con coste de nodos de forma incremental, sin reconstruirlo desde cero cada vez que varía un enlace o un peso de nodo.

De manera complementaria, podría explorarse la incorporación de elementos predictivos en los algoritmos heurísticos. Dado que los patrones de tráfico suelen exhibir picos y valles en intervalos regulares, un modelo que anticipe la demanda en ciertos tramos de la red permitiría ajustar de forma proactiva la selección de rutas y, en última instancia, el despliegue de switches.

Un tercer ámbito de mejora radica en la consideración de múltiples criterios de optimización simultáneamente. Hasta ahora, el enfoque se ha centrado en dos factores principales: el coste de enlace y el coste de nodos. Sin embargo, en la práctica existen otros elementos relevantes como puede ser la disponibilidad de enlaces redundantes o riesgo de puntos únicos de falla.

En el plano de la optimización temporal, otra línea interesante es la adaptación de los algoritmos a entornos de ingeniería de tráfico y red viva, donde la red debe optimizarse cada poco tiempo. Si bien los métodos desarrollados ofrecen buena calidad de solución, su coste computacional puede seguir siendo elevado en periodos muy cortos.

Como reflexión final, las propuestas de trabajo futuro y mejora abarcan desde la adaptación a redes dinámicas con cambios frecuentes y la incorporación de criterios de resiliencia al fallo.

5. ANEXOS

5.1. CÓDIGOS DE LOS ALGORITMOS DESARROLLADOS

Código 1: PRIM_CLÁSICO

```
import numpy as np
import os
def leer_matrices(nombre_archivo):
  with open(nombre_archivo, 'r') as archivo:
    lineas = archivo.readlines()
  matriz_costes = []
  vector_pesos = []
  seccion = None
  for linea in lineas:
    linea = linea.strip()
    if "Matriz de Adyacencia" in linea:
      seccion = "adyacencia"
      continue
    elif "Matriz de Costes" in linea:
      seccion = "costes"
      continue
    elif "Vector de Pesos" in linea:
      seccion = "pesos"
      continue
    if seccion == "costes" and linea:
      matriz_costes.append(list(map(int, linea.split())))
    elif seccion == "pesos" and linea:
      vector_pesos = list(map(int, linea.split()))
  matriz_costes = np.array(matriz_costes)
  return matriz_costes, np.array(vector_pesos)
def prim(matriz_costes):
  n = len(matriz_costes)
  visitados = [False] * n
  arbol = []
  visitados[0] = True
  num_aristas = 0
  coste_total = 0
  while num_aristas < n - 1:
    minimo = float('inf')
    nodo1 = nodo2 = -1
    for i in range(n):
      if visitados[i]:
        for j in range(n):
          if not visitados[j] and 0 < matriz_costes[i][j] < minimo:
            minimo = matriz_costes[i][j]
            nodo1, nodo2 = i, j
    if nodo1 != -1 and nodo2 != -1:
      arbol.append((nodo1 + 1, nodo2 + 1, minimo))
      visitados[nodo2] = True
      coste_total += minimo
```

```
        num_aristas += 1
    return arbol, coste_total
def nodos_conexiones(arbol):
    conexiones = {}
    for nodo1, nodo2, _ in arbol:
        conexiones[nodo1] = conexiones.get(nodo1, 0) + 1
        conexiones[nodo2] = conexiones.get(nodo2, 0) + 1
    nodos_conectados = {nodo: conexiones[nodo] for nodo in conexiones if conexiones[nodo]
>= 3}
    return nodos_conectados
def calcular_coste_nodos(arbol, nodos_conectados, vector_pesos):
    coste_total_nodos = 0
    for nodo in nodos_conectados:
        coste_total_nodos += vector_pesos[nodo - 1]
    return coste_total_nodos
def guardar_resultado(arbol, coste_total, nodos_conectados, coste_nodos, suma_total,
nombre_archivo):
    with open(nombre_archivo, 'w') as archivo:
        archivo.write("Árbol de Expansión Mínima (Prim):\n")
        archivo.write(f"Coste Total del Árbol: {coste_total}\n")
        archivo.write("Aristas: ")
        archivo.write(", ".join([f"({n1}-{n2}, {coste})" for n1, n2, coste in arbol]) + "\n")
        nodos_lista = sorted(nodos_conectados.keys())
        archivo.write(f"Nodos con tres o más conexiones: {nodos_lista}\n")
        archivo.write(f"Coste de los nodos: {coste_nodos}\n")
        archivo.write(f"Coste total: {suma_total}\n")
    print(f"Resultado guardado en: {nombre_archivo}")
def procesar_archivo_prim(ruta_archivos, ruta_salida):
    try:
        archivos = [f for f in os.listdir(ruta_archivos) if '_MATRICES' in f]
        for archivo in archivos:
            ruta_completa = os.path.join(ruta_archivos, archivo)
            matriz_costes, vector_pesos = leer_matrices(ruta_completa)
            arbol, coste_total = prim(matriz_costes)
            nodos_conectados = nodos_conexiones(arbol)
            coste_nodos = calcular_coste_nodos(arbol, nodos_conectados, vector_pesos)
            suma_total = coste_total + coste_nodos
            nombre_base = os.path.basename(archivo).replace('_MATRICES', '_RED.txt')
            nombre_salida = os.path.join(ruta_salida, nombre_base)
            guardar_resultado(arbol, coste_total, nodos_conectados, coste_nodos, suma_total,
nombre_salida)
    except Exception as e:
        print(f"Error al procesar los archivos en la ruta {ruta_archivos}: {e}")
ruta_archivos = r"C:\\Users\\juanf\\Desktop\\spd\\resultados\\Ficheros con costes"
ruta_salida = r"C:\\Users\\juanf\\Desktop\\spd\\resultados\\Ficheros con costes"
procesar_archivo_prim(ruta_archivos, ruta_salida)
```

Código 2: PRIM_DINÁMICO

```
import numpy as np
import os

def leer_matrices(nombre_archivo):
  with open(nombre_archivo, 'r') as archivo:
    lineas = archivo.readlines()[1:]
  aristas=[]
  pesos={}
  seccion=None
  for linea in lineas:
    linea=linea.strip()
    if not linea: continue
    if "Pesos" in linea: seccion="pesos"; continue
    elif seccion is None: seccion="aristas"
    if seccion=="aristas" and len(linea.split())==3:
      a,b,c=map(int,linea.split()); aristas.append((a,b,c))
    elif seccion=="pesos" and len(linea.split())==2:
      nodo,peso=map(int,linea.split()); pesos[nodo]=peso
  n=max(pesos.keys())
  matriz_costes=np.zeros((n,n),dtype=int)
  for a,b,c in aristas:
    matriz_costes[a-1][b-1]=c; matriz_costes[b-1][a-1]=c
  vector_pesos=np.array([pesos[i] for i in range(1,n+1)])
  return matriz_costes,vector_pesos

def prim_dinamico(matriz_costes,vector_pesos):
  n=len(matriz_costes)
  visited=[False]*n
  degrees=[0]*n
  arbol=[]
  visited[0]=True
  num_aristas=0
  total_aristas=0
  total_nodos=0
  while num_aristas<n-1:
    minimo=float('inf')
    candidato=None
    for i in range(n):
      if visited[i]:
        for j in range(n):
          if not visited[j] and matriz_costes[i][j]>0:
            coste_original=matriz_costes[i][j]
            coste_modificado=coste_original
            if degrees[i]==2: coste_modificado+=vector_pesos[i]
            if coste_modificado<minimo:
              minimo=coste_modificado
              candidato=(i,j,coste_original,vector_pesos[i] if degrees[i]==2 else 0)
    if candidato is None: break
    i,j,coste_original,coste_extra=candidato
    arbol.append((i+1,j+1,coste_original,coste_extra))
```

```
      visited[j]=True
      degrees[i]+=1; degrees[j]+=1
      total_aristas+=coste_original
      total_nodos+=coste_extra
      num_aristas+=1
  return arbol,total_aristas,degrees

def nodos_conexiones(degrees):
  return {idx+1:grado for idx,grado in enumerate(degrees) if grado>=3}

def calcular_coste_nodos(arbol,nodos_conectados,vector_pesos):
  return sum(vector_pesos[nodo-1] for nodo in nodos_conectados)

def
guardar_resultado(arbol,total_aristas,nodos_conectados,coste_nodos,suma_total,nombre_a
rchivo):
  with open(nombre_archivo,'w') as archivo:
    archivo.write("Árbol de Expansión Mínima (Prim dinámico):\n")
    aristas_str=[]
    for n1,n2,coste_original,coste_extra in arbol:
      coste_mostrado=coste_original+coste_extra
      if coste_extra>0:
        aristas_str.append(f"({n1}-{n2}, {coste_original}+{coste_extra}={coste_mostrado})")
      else:
        aristas_str.append(f"({n1}-{n2}, {coste_mostrado})")
    archivo.write("Aristas: "+", ".join(aristas_str)+"\n")
    nodos_lista=sorted(nodos_conectados.keys())
    archivo.write(f"Nodos con tres o más conexiones: {nodos_lista}\n")
    archivo.write(f"Coste Total del Árbol: {total_aristas}\n")
    archivo.write(f"Coste de los nodos: {coste_nodos}\n")
    archivo.write(f"Suma total: {suma_total}\n")
  print(f"Resultado guardado en: {nombre_archivo}")

def procesar_archivo_prim_dinamico(ruta_archivos,ruta_salida):
  try:
    archivos=[f for f in os.listdir(ruta_archivos) if f.endswith("_RED.txt")]1
    print(f"Archivos encontrados: {archivos}")
    for archivo in archivos:
      ruta_completa=os.path.join(ruta_archivos,archivo)
      matriz_costes,vector_pesos=leer_matrices(ruta_completa)
      arbol,total_aristas,degrees=prim_dinamico(matriz_costes,vector_pesos)
      nodos_conectados=nodos_conexiones(degrees)
      coste_nodos=calcular_coste_nodos(arbol,nodos_conectados,vector_pesos)
      suma_total=total_aristas+coste_nodos
      nombre_base=os.path.basename(archivo).replace('_RES','_DINAMICO')
      nombre_salida=os.path.join(ruta_salida,nombre_base)

guardar_resultado(arbol,total_aristas,nodos_conectados,coste_nodos,suma_total,nombre_s
alida)
  except Exception as e:
    print(f"Error al procesar los archivos en la ruta {ruta_archivos}: {e}")
```

```
ruta_archivos=r"C:\\Users\\juanf\\Desktop\\spd\\resultados\\Ficheros con costes"
ruta_salida=r"C:\\Users\\juanf\\Desktop\\spd\\resultados\\Ficheros con costes"
procesar_archivo_prim_dinamico(ruta_archivos,ruta_salida)
```

Código 3: HEURÍSTICO N1

```
import numpy as np
import os
import pandas as pd
def read_res_file(file_path):
  with open(file_path,'r') as f:
    lines=[line.strip() for line in f.readlines() if line.strip()!='']
  parts=lines[0].split()
  n=int(parts[0]);m=int(parts[1])
  edges=[]
  for line in lines[1:1+m]:
    p=line.split();u=int(p[0]);v=int(p[1]);cost=float(p[2]);edges.append((u,v,cost))
  peso_index=next(i for i,line in enumerate(lines) if line.startswith("Pesos de los nodos"))
  node_weights=[0]*n
  for line in lines[peso_index+1:]:
    p=line.split();node=int(p[0]);weight=float(p[1]);node_weights[node-1]=weight
  return n,edges,node_weights
def build_cost_matrix(n,edges):
  matrix=np.zeros((n,n))
  for u,v,cost in edges:
    matrix[u-1,v-1]=cost;matrix[v-1,u-1]=cost
  return matrix
def prim(matrix):
  n=len(matrix)
  visited=[False]*n;visited[0]=True
  tree_edges=[];num_edges=0;total_cost=0.0
  while num_edges<n-1:
    min_cost=float('inf');u_sel=v_sel=-1
    for i in range(n):
      if visited[i]:
        for j in range(n):
          if not visited[j] and 0<matrix[i][j]<min_cost:
            min_cost=matrix[i][j];u_sel=i;v_sel=j
    if u_sel!=-1:
      tree_edges.append((u_sel+1,v_sel+1,min_cost))
      visited[v_sel]=True;total_cost+=min_cost;num_edges+=1
  return tree_edges,total_cost
def compute_degrees(tree_edges):
  degrees={}
  for u,v,_ in tree_edges:
    degrees[u]=degrees.get(u,0)+1;degrees[v]=degrees.get(v,0)+1
  return degrees
def modify_cost_matrix(n,original_matrix,mst_classic,switches,extra):
  modified_matrix=np.copy(original_matrix)
  mst_edges_set={(min(u,v),max(u,v)) for u,v,_ in mst_classic}
  for u,v in mst_edges_set:
    i=u-1; j=v-1; supplement=0.0
    if u in switches: supplement+=extra[u]
    if v in switches: supplement+=extra[v]
    modified_matrix[i,j]=original_matrix[i,j]+supplement
    modified_matrix[j,i]=original_matrix[j,i]+supplement
```

```
    return modified_matrix,mst_edges_set
def revert_mst_to_original(mst_heuristic,original_matrix):
    final_mst=[];final_total_cost=0.0
    for u,v,_ in mst_heuristic:
        cost_original=original_matrix[u-1,v-1]
        final_mst.append((u,v,cost_original));final_total_cost+=cost_original
    return final_mst,final_total_cost
def procesar_archivo_heuristico_info(res_file_path):
    n,edges,node_weights=read_res_file(res_file_path)
    original_matrix=build_cost_matrix(n,edges)
    mst_classic,total_classic=prim(original_matrix)
    degrees_classic=compute_degrees(mst_classic)
    switches=[node for node,deg in degrees_classic.items() if deg>=3]
    extra={node:node_weights[node-1]/degrees_classic[node] for node in switches}
    modified_matrix,_=modify_cost_matrix(n,original_matrix,mst_classic,switches,extra)
    mst_heuristic,total_heuristic=prim(modified_matrix)
    degrees_heuristic=compute_degrees(mst_heuristic)
    final_mst,final_total_cost=revert_mst_to_original(mst_heuristic,original_matrix)
    final_switches=sorted([node for node,deg in degrees_heuristic.items() if deg>=3])
    coste_nodos_final=sum(node_weights[node-1] for node in final_switches)
    coste_total=final_total_cost+coste_nodos_final
    summary={
        'Fichero':os.path.basename(res_file_path),
        'Coste Total del Árbol (aristas)':final_total_cost,
        'Coste de los nodos (switches finales)':coste_nodos_final,
        'Coste Total (aristas + nodos)':coste_total,
        'Nodos finales (>=3 conexiones)':str(final_switches)
    }
    edges_info=[]
    for u,v,cost in final_mst:
        edges_info.append({'Fichero':os.path.basename(res_file_path),'Nodo U':u,'Nodo
V':v,'Coste':cost})
    return summary,edges_info
def procesar_directorio_heuristico(directorio,output_excel):
    archivos=[f for f in os.listdir(directorio) if f.endswith('_RED.txt')]
    if not archivos: print("No se encontraron archivos _RED.txt en el directorio.");return
    lista_resumen=[];lista_aristas=[]
    for archivo in archivos:
        ruta=os.path.join(directorio,archivo)
        try:
            summary,edges_info=procesar_archivo_heuristico_info(ruta)
            lista_resumen.append(summary);lista_aristas.extend(edges_info)
        except Exception as e:
            print(f"Error al procesar {archivo}: {e}")
    df_resumen=pd.DataFrame(lista_resumen)
    df_aristas=pd.DataFrame(lista_aristas)
    with pd.ExcelWriter(output_excel,engine='openpyxl') as writer:
        df_resumen.to_excel(writer,sheet_name='Resumen',index=False)
        df_aristas.to_excel(writer,sheet_name='Aristas',index=False)
    print(f"Resultado heurístico consolidado guardado en: {output_excel}")
if __name__=='__main__':
    directorio=r"C:\Users\juanf\Desktop\spd\resultados\Ficheros con costes"
```

```
output_excel=os.path.join(directorio,"Resultado_Heuristico_Consolidado.xlsx")
procesar_directorio_heuristico(directorio,output_excel)
```

Código 4: HEURÍSTICO N2

```
import numpy as np
import os
import pandas as pd

def read_red_file(file_path):
  with open(file_path,'r') as f:
    lines=[l.strip() for l in f.readlines() if l.strip()]
  parts=lines[0].split()
  n=int(parts[0]);m=int(parts[1])
  edges=[]
  for line in lines[1:1+m]:
    u,v,cost=line.split();edges.append((int(u),int(v),float(cost)))
  peso_index=next(i for i,ln in enumerate(lines) if ln.startswith("Pesos de los nodos"))
  node_weights=[0]*n
  for ln in lines[peso_index+1:]:
    node,weight=ln.split();node_weights[int(node)-1]=float(weight)
  return n,edges,node_weights

def build_cost_matrix(n,edges):
  mat=np.zeros((n,n))
  for u,v,c in edges:
    mat[u-1,v-1]=c;mat[v-1,u-1]=c
  return mat

def prim(mat):
  n=len(mat);visited=[False]*n;visited[0]=True
  tree_edges=[];total_cost=0.0;num_edges=0
  while num_edges<n-1:
    min_cost=float('inf');u_sel=v_sel=-1
    for i in range(n):
      if visited[i]:
        for j in range(n):
          if not visited[j] and mat[i,j]>0 and mat[i,j]<min_cost:
            min_cost=mat[i,j];u_sel=i;v_sel=j
    if u_sel>=0:
      tree_edges.append((u_sel+1,v_sel+1,min_cost))
      visited[v_sel]=True
      total_cost+=min_cost;num_edges+=1
  return tree_edges,total_cost

def compute_graph_degrees(edges,n):
  deg={i:0 for i in range(1,n+1)}
  for u,v,_ in edges:
    deg[u]+=1;deg[v]+=1
  return deg

def modify_cost_matrix_graph(n,orig,sw,extra):
  mod=orig.copy()
  for i in range(n):
    for j in range(n):
```

```
            if orig[i,j]>0:
                s=0.0
                if i+1 in sw: s+=extra[i+1]
                if j+1 in sw: s+=extra[j+1]
                mod[i,j]=orig[i,j]+s
    return mod

def revert_mst_to_original(mst,orig):
    final=[];tc=0.0
    for u,v,_ in mst:
        c=orig[u-1,v-1]
        final.append((u,v,c));tc+=c
    return final,tc

def procesar_archivo_heuristico_v2_info(red_file_path):
    n,edges,node_weights=read_red_file(red_file_path)
    orig=build_cost_matrix(n,edges)
    deg=compute_graph_degrees(edges,n)
    switches=[node for node,d in deg.items() if d>=3]
    extra={node:node_weights[node-1] for node in switches}
    mod=modify_cost_matrix_graph(n,orig,switches,extra)
    mst_h,_=prim(mod)
    final_mst,mst_cost=revert_mst_to_original(mst_h,orig)
    deg_final={}
    for u,v,_ in final_mst:
        deg_final[u]=deg_final.get(u,0)+1;deg_final[v]=deg_final.get(v,0)+1
    final_switches=[n for n,d in deg_final.items() if d>=3 and n in switches]
    penalty=sum(extra[n] for n in final_switches)
    total=mst_cost+penalty
    summary={'Fichero':os.path.basename(red_file_path),'Coste
Árbol':mst_cost,'Penalización':penalty,'Coste Total':total,'Switches':sorted(final_switches)}
    edges_info=[{'Fichero':os.path.basename(red_file_path),'Nodo U':u,'Nodo V':v,'Coste':c} for
u,v,c in final_mst]
    return summary,edges_info

def procesar_directorio_heuristico_v2(dir_path,output_excel):
    archivos=[f for f in os.listdir(dir_path) if f.endswith('_RED.txt')]
    if not archivos:
        print("No se encontraron archivos _RED.txt")
        return
    res=[];aristas=[]
    for a in archivos:
        path=os.path.join(dir_path,a)
        try:
            s,ei=procesar_archivo_heuristico_v2_info(path)
            res.append(s);aristas.extend(ei)
        except Exception as e:
            print(f"Error en {a}: {e}")
    df_res=pd.DataFrame(res)
    df_edges=pd.DataFrame(aristas)
    with pd.ExcelWriter(output_excel,engine='openpyxl') as w:
        df_res.to_excel(w,sheet_name='Resumen',index=False)
```

```
        df_edges.to_excel(w,sheet_name='Aristas',index=False)
    print(f"Heurístico V2 consolidado guardado en: {output_excel}")

if __name__=='__main__':
    directorio=r"C:\Users\juanf\Desktop\spd\resultados\Ficheros con costes"
    output_excel=os.path.join(directorio,"Resultado_Heuristico_V2_Consolidado.xlsx")
    procesar_directorio_heuristico_v2(directorio,output_excel)
```

Código 5: HEURÍSTICO N3

```
import numpy as np
import os
import pandas as pd
from tqdm import tqdm
import time
from concurrent.futures import ProcessPoolExecutor, as_completed
from scipy.sparse.csgraph import minimum_spanning_tree
def read_red_file(file_path):
  with open(file_path, 'r') as f:
    lines = [line.strip() for line in f.readlines() if line.strip()]
  parts = lines[0].split()
  n = int(parts[0]); m = int(parts[1])
  edges = [(int(parts[0]), int(parts[1]), float(parts[2])) for parts in (line.split() for line in
lines[1:1+m])]
  peso_index = next(i for i,line in enumerate(lines) if line.startswith("Pesos de los nodos"))
  node_weights = [0]*n
  for line in lines[peso_index+1:]:
    parts=line.split(); node_weights[int(parts[0])-1]=float(parts[1])
  return n, edges, node_weights
def build_cost_matrix(n, edges):
  matrix = np.full((n, n), np.inf)
  np.fill_diagonal(matrix, 0)
  for u,v,cost in edges:
    matrix[u-1,v-1]=cost; matrix[v-1,u-1]=cost
  return matrix
def prim_scipy(matrix):

mst_sparse=minimum_spanning_tree(matrix);mst_dense=mst_sparse.toarray();n=matrix.sha
pe[0];tree_edges=[];total_cost=0.0
  for i in range(n):
    for j in range(n):
      if i!=j and mst_dense[i,j] and mst_dense[i,j]!=np.inf:
        tree_edges.append((i+1,j+1,mst_dense[i,j]));total_cost+=mst_dense[i,j]
  return tree_edges,total_cost
def compute_degrees(edges):
  degrees={}
  for u,v,_ in edges:
    degrees[u]=degrees.get(u,0)+1;degrees[v]=degrees.get(v,0)+1
  return degrees
def revert_mst_to_original(mst_heuristic, original_matrix):
  final_mst=[];total=0.0
  for u,v,_ in mst_heuristic:
    cost_orig=original_matrix[u-1][v-1]
    final_mst.append((u,v,cost_orig));total+=cost_orig
  return final_mst,total
def procesar_archivo_heuristico_v3_info(red_file_path,iterations=500):
  n,edges,node_weights=read_red_file(red_file_path)
  original_matrix=build_cost_matrix(n,edges)
  mst_classic,_=prim_scipy(original_matrix)
  degrees_classic=compute_degrees(mst_classic)
```

```
  switches=[node for node,deg in degrees_classic.items() if deg>=3]
  switches_set=set(switches)
  incident_edges={node:[] for node in switches}
  for edge in edges:
    u,v,cost_edge=edge;key=(min(u,v),max(u,v))
    if u in switches_set:incident_edges[u].append((key,cost_edge))
    if v in switches_set:incident_edges[v].append((key,cost_edge))
  edges_array=np.array(edges);u_idx=edges_array[:,0].astype(int)-
1;v_idx=edges_array[:,1].astype(int)-1;cost_edge_arr=edges_array[:,2].astype(float)
  edge_keys=[(min(u,v),max(u,v)) for u,v,_ in edges]
  edge_to_switches={}
  for sw,inc_edges in incident_edges.items():
    for key,_ in inc_edges:edge_to_switches.setdefault(key,[]).append(sw)
  best_final_cost=float('inf');best_candidate_mst=None;best_candidate_switches=None
  for _ in tqdm(range(iterations),desc=f"Iteraciones {iterations}",leave=False):
    extra_partition={}
    for node in switches:
      edges_list=incident_edges[node]
      if not edges_list:continue
      k=len(edges_list);r=np.random.rand(k)
      edge_costs=np.array([c for _,c in edges_list])
      weighted=r*edge_costs;sum_weighted=weighted.sum()
      partition=weighted/sum_weighted if sum_weighted else np.ones(k)/k
      extras=partition*node_weights[node-1];extra_partition[node]={}
      for i,(key,_) in enumerate(edges_list):extra_partition[node][key]=extras[i]
    modified_matrix=original_matrix.copy()
    extra_values=np.array([sum(extra_partition.get(sw,{}).get(key,0.0) for sw in
edge_to_switches.get(key,[])) for key in edge_keys])

modified_matrix[u_idx,v_idx]=cost_edge_arr+extra_values;modified_matrix[v_idx,u_idx]=cost_
edge_arr+extra_values
    candidate_mst,_=prim_scipy(modified_matrix)

final_candidate_mst,candidate_mst_cost=revert_mst_to_original(candidate_mst,original_mat
rix)
    candidate_degrees={}
    for u,v,_ in final_candidate_mst:

candidate_degrees[u]=candidate_degrees.get(u,0)+1;candidate_degrees[v]=candidate_degre
es.get(v,0)+1
    candidate_switches=[node for node,deg in candidate_degrees.items() if deg>=3 and node
in switches_set]
    candidate_penalty=sum(node_weights[node-1] for node in candidate_switches)
    candidate_final_cost=candidate_mst_cost+candidate_penalty
    if candidate_final_cost<best_final_cost:

best_final_cost=candidate_final_cost;best_candidate_mst=final_candidate_mst;best_candid
ate_switches=candidate_switches
  summary={'Fichero':os.path.basename(red_file_path),'Coste Total del Árbol
(aristas)':sum(e[2] for e in best_candidate_mst),'Coste de los nodos
(penalización)':sum(node_weights[node-1] for node in best_candidate_switches),'Coste Total
```

```
(aristas + nodos)':best_final_cost,'Nodos finales (>=3 conexiones y
switches)':str(sorted(best_candidate_switches))}
  edges_info=[{'Fichero':os.path.basename(red_file_path),'Nodo U':u,'Nodo V':v,'Coste':cost}
for u,v,cost in best_candidate_mst]
  return summary,edges_info
def process_file_for_iterations(file_path,iteration_values):
  summaries=[];edges_infos=[]
  for iter_val in iteration_values:
    summary,edges_info=procesar_archivo_heuristico_v3_info(file_path,iterations=iter_val)
    summary['Iteraciones']=iter_val;summaries.append(summary)
    for edge in edges_info:edge['Iteraciones']=iter_val;edges_infos.append(edge)
  return summaries,edges_infos
def
procesar_directorio_heuristico_v3_iterations_parallel(directorio,output_excel,iteration_value
s=[1,10,50,100,500],max_workers=None):
  archivos=[os.path.join(directorio,f) for f in os.listdir(directorio) if f.endswith('_RED.txt')]
  if not archivos:print("No se encontraron archivos _RED.txt en el directorio
especificado.");return
  all_summaries=[];all_edges=[];start_time=time.time()
  with ProcessPoolExecutor(max_workers=max_workers) as executor:
    futures={executor.submit(process_file_for_iterations,file_path,iteration_values):file_path
for file_path in archivos}
    for future in tqdm(as_completed(futures),total=len(futures),desc="Ficheros
procesados"):
      try:

summaries,edges_infos=future.result();all_summaries.extend(summaries);all_edges.extend(
edges_infos)
      except Exception as e:print("Error procesando un fichero:",e)
  elapsed=time.time()-start_time;print(f"Tiempo total de ejecución: {elapsed:.2f} segundos")
  df_resumen=pd.DataFrame(all_summaries);df_aristas=pd.DataFrame(all_edges)
  with pd.ExcelWriter(output_excel,engine='openpyxl') as writer:

df_resumen.to_excel(writer,sheet_name='Resumen',index=False);df_aristas.to_excel(writer,s
heet_name='Aristas',index=False)
  print(f"Resultado consolidado guardado en: {output_excel}")
if __name__=='__main__':
  directorio=r"C:\Users\juanf\Desktop\spd\resultados\Ficheros con costes"
  output_excel=os.path.join(directorio,"Resultado_Heuristico_V3.xlsx")
  iteraciones_valores=[1,10,50,100,500]

procesar_directorio_heuristico_v3_iterations_parallel(directorio,output_excel,iteration_value
s=iteraciones_valores,max_workers=4)
```

Código 6: GENÉTICO

```
import numpy as np
import os
import pandas as pd
import time
from tqdm import tqdm
from concurrent.futures import ProcessPoolExecutor, as_completed
from scipy.sparse.csgraph import minimum_spanning_tree
def read_res_file(file_path):
  """
  Lee el archivo _RED.txt y extrae:
   - n: número de nodos
   - edges: lista de aristas (u, v, coste)
   - node_weights: lista de pesos de cada nodo
  """
  with open(file_path, 'r') as f:
    lines = f.readlines()
  lines = [line.strip() for line in lines if line.strip()!='']
  parts = lines[0].split()
  n = int(parts[0])
  m = int(parts[1])
  edges = []
  for line in lines[1:1+m]:
    parts = line.split()
    u = int(parts[0])
    v = int(parts[1])
    cost = float(parts[2])
    edges.append((u, v, cost))
  peso_index = None
  for i, line in enumerate(lines):
    if line.startswith("Pesos de los nodos"):
      peso_index = i
      break
  node_weights = [0] * n
  for line in lines[peso_index+1:]:
    parts = line.split()
    node = int(parts[0])
    weight = float(parts[1])
    node_weights[node-1] = weight
  return n, edges, node_weights
def build_cost_matrix(n, edges):
  matrix = np.full((n, n), np.inf)
  np.fill_diagonal(matrix, 0)
  for u, v, cost in edges:
    matrix[u-1, v-1] = cost
    matrix[v-1, u-1] = cost
  return matrix
def prim_scipy(matrix):
  mst_sparse = minimum_spanning_tree(matrix)
  mst_dense = mst_sparse.toarray()
  n = matrix.shape[0]
```

```
    tree_edges = []
    total_cost = 0.0
    for i in range(n):
        for j in range(n):
            if i != j and mst_dense[i, j] != 0 and mst_dense[i, j] != np.inf:
                tree_edges.append((i+1, j+1, mst_dense[i, j]))
                total_cost += mst_dense[i, j]
    return tree_edges, total_cost
def compute_degrees(edges):
    degrees = {}
    for u, v, _ in edges:
        degrees[u] = degrees.get(u, 0) + 1
        degrees[v] = degrees.get(v, 0) + 1
    return degrees
def revert_mst_to_original(mst_candidate, original_matrix):
    final_mst = []
    total = 0.0
    for u, v, _ in mst_candidate:
        cost_orig = original_matrix[u-1][v-1]
        final_mst.append((u, v, cost_orig))
        total += cost_orig
    return final_mst, total
def generate_candidate_genome(switches, incident_edges, node_weights):
    genome = {}
    for sw in switches:
        edges_list = incident_edges[sw]
        if not edges_list:
            continue
        k = len(edges_list)
        r = np.random.rand(k)
        base_costs = np.array([c for (_, c) in edges_list])
        weighted = r * base_costs
        sum_weighted = np.sum(weighted)
        partition = (weighted / sum_weighted) if sum_weighted != 0 else np.ones(k)/k
        extras = partition * node_weights[sw-1]
        genome[sw] = extras
    return genome
def evaluate_genome(genome, original_matrix, cost_edge_arr, u_idx, v_idx, edge_keys,
edge_to_switches, mapping, node_weights, switches):
    extra_values = np.array([
        sum(genome.get(sw, [0])[mapping[(sw, key)]] for sw in edge_to_switches.get(key, []))
        for key in edge_keys
    ])
    modified_matrix = original_matrix.copy()
    modified_matrix[u_idx, v_idx] = cost_edge_arr + extra_values
    modified_matrix[v_idx, u_idx] = cost_edge_arr + extra_values
    candidate_mst, _ = prim_scipy(modified_matrix)
    final_candidate_mst, candidate_mst_cost = revert_mst_to_original(candidate_mst,
original_matrix)
    candidate_degrees = {}
    for u, v, _ in final_candidate_mst:
        candidate_degrees[u] = candidate_degrees.get(u, 0) + 1
```

```
        candidate_degrees[v] = candidate_degrees.get(v, 0) + 1
    candidate_switches = [node for node, deg in candidate_degrees.items() if deg >= 3 and node
in set(switches)]
    candidate_penalty = sum(node_weights[node-1] for node in candidate_switches)
    candidate_final_cost = candidate_mst_cost + candidate_penalty
    return candidate_final_cost, candidate_mst_cost, candidate_penalty, final_candidate_mst
def generate_candidate(switches, incident_edges, node_weights, original_matrix,
cost_edge_arr, u_idx, v_idx, edge_keys, edge_to_switches, mapping):
    genome = generate_candidate_genome(switches, incident_edges, node_weights)
    final_cost, mst_cost, penalty, _ = evaluate_genome(genome, original_matrix, cost_edge_arr,
u_idx, v_idx, edge_keys, edge_to_switches, mapping, node_weights, switches)
    return {'genome': genome, 'final_cost': final_cost, 'mst_cost': mst_cost, 'penalty': penalty}
def genetic_algorithm_file(res_file_path, num_initial=1000, num_mixes=5000):
    n, edges, node_weights = read_res_file(res_file_path)
    original_matrix = build_cost_matrix(n, edges)
    mst_classic, _ = prim_scipy(original_matrix)
    degrees_classic = compute_degrees(mst_classic)
    switches = [node for node, deg in degrees_classic.items() if deg >= 3]
    switches_set = set(switches)
    incident_edges = {node: [] for node in switches}
    for edge in edges:
        u, v, cost_edge = edge
        key = (min(u, v), max(u, v))
        if u in switches_set:
            incident_edges[u].append((key, cost_edge))
        if v in switches_set:
            incident_edges[v].append((key, cost_edge))
    edges_array = np.array(edges)
    u_idx = edges_array[:, 0].astype(int) - 1
    v_idx = edges_array[:, 1].astype(int) - 1
    cost_edge_arr = edges_array[:, 2].astype(float)
    edge_keys = [(min(u, v), max(u, v)) for (u, v, _) in edges]
    edge_to_switches = {}
    for sw, inc_edges in incident_edges.items():
        for key, _ in inc_edges:
            edge_to_switches.setdefault(key, []).append(sw)
    mapping = {}
    for sw, inc_edges in incident_edges.items():
        for i, (key, _) in enumerate(inc_edges):
            mapping[(sw, key)] = i
    population = []
    for _ in tqdm(range(num_initial), desc="Generando población inicial", leave=False):
        cand = generate_candidate(switches, incident_edges, node_weights, original_matrix,
cost_edge_arr, u_idx, v_idx, edge_keys, edge_to_switches, mapping)
        population.append(cand)
    mix_results = []
    for mix_iter in tqdm(range(num_mixes), desc="Realizando mezclas", leave=False):
        idx1, idx2 = np.random.choice(len(population), size=2, replace=False)
        parent1 = population[idx1]
        parent2 = population[idx2]
        child_genome = {}
        for sw in switches:
```

```
        vec1 = parent1['genome'][sw]
        vec2 = parent2['genome'][sw]
        child_genome[sw] = (vec1 + vec2) / 2.0
      child_final_cost, child_mst_cost, child_pen, _ = evaluate_genome(child_genome,
original_matrix, cost_edge_arr, u_idx, v_idx, edge_keys, edge_to_switches, mapping,
node_weights, switches)
      if child_final_cost < max(parent1['final_cost'], parent2['final_cost']):
        if parent1['final_cost'] >= parent2['final_cost']:
          population[idx1] = {'genome': child_genome, 'final_cost': child_final_cost, 'mst_cost':
child_mst_cost, 'penalty': child_pen}
          replaced_idx = idx1
        else:
          population[idx2] = {'genome': child_genome, 'final_cost': child_final_cost, 'mst_cost':
child_mst_cost, 'penalty': child_pen}
          replaced_idx = idx2
        replaced = True
      else:
        replaced = False
      mix_results.append({'Mix Iteración': mix_iter+1, 'Parent1_Costo': parent1['final_cost'],
'Parent2_Costo': parent2['final_cost'], 'Child_Costo': child_final_cost, 'Reemplazo': replaced})
  best_candidate = min(population, key=lambda c: c['final_cost'])
  return mix_results, best_candidate
def process_file_genetic(file_path, num_initial=1000, num_mixes=5000):
  mix_results, best_candidate = genetic_algorithm_file(file_path, num_initial, num_mixes)
  filename = os.path.basename(file_path)
  for record in mix_results:
    record['Fichero'] = filename
  resumen = {'Fichero': filename, 'Mejor_Costo_Final': best_candidate['final_cost'],
'Mejor_MST_Costo': best_candidate['mst_cost'], 'Mejor_Penalización':
best_candidate['penalty']}
  return filename, mix_results, resumen
def procesar_directorio_genetico_parallel(directorio, output_excel, num_initial=1000,
num_mixes=5000, max_workers=None):
  archivos = [os.path.join(directorio, f) for f in os.listdir(directorio) if f.endswith('_RED.txt')]
  if not archivos:
    print("No se encontraron archivos _RED.txt en el directorio especificado.")
    return
  hojas = {}
  resumenes = []
  start_time = time.time()
  with ProcessPoolExecutor(max_workers=max_workers) as executor:
    futures = {executor.submit(process_file_genetic, fp, num_initial, num_mixes): fp for fp in
archivos}
    for future in tqdm(as_completed(futures), total=len(futures), desc="Procesando ficheros
(Genético)"):
      try:
        filename, mix_results, resumen = future.result()
        sheet_name = os.path.splitext(filename)[0]
        hojas[sheet_name] = pd.DataFrame(mix_results)
        resumenes.append(resumen)
      except Exception as e:
        print("Error procesando un fichero:", e)
```

```
    elapsed = time.time() - start_time
    print(f"Tiempo total: {elapsed:.2f} segundos")
    with pd.ExcelWriter(output_excel, engine='openpyxl') as writer:
        for sheet_name, df in hojas.items():
            df.to_excel(writer, sheet_name=sheet_name, index=False)
        pd.DataFrame(resumenes).to_excel(writer, sheet_name='Resumen', index=False)
    print(f"Resultados guardados en: {output_excel}")
if __name__=="__main__":
    directorio = r"C:\Users\juanf\Desktop\spd\resultados\temporal"
    output_excel = os.path.join(directorio, "Resultado_genetico.xlsx")
    procesar_directorio_genetico_parallel(directorio, output_excel, num_initial=1000,
num_mixes=5000, max_workers=4)
```

5.2. TABLAS DE RESULTADOS

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_20_27_211	334	173	381	207	139	140
Spd_RF2_20_27_219	351	300	358	285	192	189
Spd_RF2_20_27_227	231	305	388	261	162	162
Spd_RF2_20_27_235	473	342	400	343	256	256
Spd_RF2_20_27_243	329	244	449	329	242	242
Spd_RF2_20_34_251	319	321	282	319	84	84
Spd_RF2_20_34_259	278	172	317	202	98	98
Spd_RF2_20_34_267	340	252	221	213	127	126
Spd_RF2_20_34_275	440	245	402	279	176	164
Spd_RF2_20_34_283	483	237	422	295	180	182
Spd_RF2_20_42_291	510	291	420	317	155	124
Spd_RF2_20_42_299	399	212	356	281	65	64
Spd_RF2_20_42_307	394	224	390	242	75	75
Spd_RF2_20_42_315	285	190	390	218	86	92
Spd_RF2_20_42_323	208	95	347	221	78	78
Spd_RF2_20_49_331	262	134	462	200	67	67
Spd_RF2_20_49_339	268	61	449	223	55	54
Spd_RF2_20_49_347	412	138	395	229	83	80
Spd_RF2_20_49_355	309	85	281	154	70	69
Spd_RF2_20_49_363	351	75	516	178	68	68
Spd_RF2_20_57_371	306	204	420	191	44	46
Spd_RF2_20_57_379	460	101	416	259	59	60
Spd_RF2_20_57_387	300	149	408	116	83	83
Spd_RF2_20_57_395	136	86	207	181	55	55
Spd_RF2_20_57_403	178	63	267	214	52	51

Tabla 6. Coste total obtenido en redes con 20 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_40_50_611	944	870	1212	830	736	735
Spd_RF2_40_50_619	686	627	652	542	507	507
Spd_RF2_40_50_627	823	812	939	681	608	608
Spd_RF2_40_50_635	1004	834	1034	783	692	691
Spd_RF2_40_50_643	711	711	782	638	562	533
Spd_RF2_40_60_651	829	684	976	591	417	415
Spd_RF2_40_60_659	725	593	757	598	304	303
Spd_RF2_40_60_667	841	672	897	625	374	369
Spd_RF2_40_60_675	803	544	914	417	376	374
Spd_RF2_40_60_683	850	478	797	513	295	332
Spd_RF2_40_71_691	595	623	526	600	183	182
Spd_RF2_40_71_699	837	467	630	414	218	219
Spd_RF2_40_71_707	772	531	789	661	289	270
Spd_RF2_40_71_715	733	452	859	552	270	269
Spd_RF2_40_71_723	597	371	874	468	180	177
Spd_RF2_40_81_731	913	500	587	534	182	182
Spd_RF2_40_81_739	890	407	717	539	220	218
Spd_RF2_40_81_747	627	355	608	442	166	164
Spd_RF2_40_81_755	668	419	747	431	150	146
Spd_RF2_40_81_763	732	363	883	414	196	180
Spd_RF2_40_92_771	723	394	610	406	155	149
Spd_RF2_40_92_779	540	278	590	309	132	130
Spd_RF2_40_92_787	680	403	607	409	172	167
Spd_RF2_40_92_795	733	303	890	404	171	169
Spd_RF2_40_92_803	613	408	752	403	167	166

Tabla 7. Coste total obtenido en redes con 40 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_60_71_1011	1297	830	989	909	717	717
Spd_RF2_60_71_1019	1246	1252	1365	1197	1037	1037
Spd_RF2_60_71_1027	1143	1195	1228	1094	958	958
Spd_RF2_60_71_1035	1218	1014	980	1000	847	847
Spd_RF2_60_71_1043	1536	1366	1302	1289	1222	1216
Spd_RF2_60_83_1051	999	979	1345	1013	563	568
Spd_RF2_60_83_1059	1187	833	1082	952	658	657
Spd_RF2_60_83_1067	1295	966	1103	1070	589	560
Spd_RF2_60_83_1075	1395	1296	1305	1032	873	846
Spd_RF2_60_83_1083	1258	914	1360	963	703	679
Spd_RF2_60_95_1091	1386	956	1499	1106	708	692
Spd_RF2_60_95_1099	1021	710	1224	685	447	453
Spd_RF2_60_95_1107	1236	912	1283	903	488	457
Spd_RF2_60_95_1115	1283	1100	1486	1176	737	736
Spd_RF2_60_95_1123	1105	984	1463	1096	592	533
Spd_RF2_60_107_1131	1401	892	1360	865	474	372
Spd_RF2_60_107_1139	1301	948	1360	690	462	459
Spd_RF2_60_107_1147	1188	835	1174	996	305	296
Spd_RF2_60_107_1155	1141	825	1570	891	432	438
Spd_RF2_60_107_1163	771	709	1085	730	339	337
Spd_RF2_60_119_1171	1133	692	1100	801	319	324
Spd_RF2_60_119_1179	1014	859	1199	742	442	410
Spd_RF2_60_119_1187	1336	612	1236	913	350	341
Spd_RF2_60_119_1195	883	512	1116	685	240	232
Spd_RF2_60_119_1203	1009	555	1052	696	325	330

Tabla 8. Coste total obtenido en redes con 60 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_80_93_1411	1686	1306	1413	1382	1219	1214
Spd_RF2_80_93_1419	1398	1359	1375	1304	1031	970
Spd_RF2_80_93_1427	2093	1875	1957	1894	1616	1614
Spd_RF2_80_93_1435	1785	1641	1587	1476	1386	1357
Spd_RF2_80_93_1443	1733	1687	1780	1706	1300	1298
Spd_RF2_80_106_1451	1680	1411	1684	1534	1006	961
Spd_RF2_80_106_1459	1302	1007	1550	1060	913	909
Spd_RF2_80_106_1467	1574	1252	1569	1398	970	974
Spd_RF2_80_106_1475	1767	1504	1745	1489	1196	1168
Spd_RF2_80_106_1483	1499	1131	1428	1191	934	926
Spd_RF2_80_120_1491	1344	1034	1537	1014	718	688
Spd_RF2_80_120_1499	1496	1058	1699	1179	797	759
Spd_RF2_80_120_1507	1999	1488	1780	1301	1032	1002
Spd_RF2_80_120_1515	1509	1181	1655	1185	836	832
Spd_RF2_80_120_1523	1676	1338	1506	1283	903	845
Spd_RF2_80_133_1531	1371	1135	1568	948	622	611
Spd_RF2_80_133_1539	1424	1019	1379	1080	584	578
Spd_RF2_80_133_1547	1438	1155	1826	1063	635	611
Spd_RF2_80_133_1555	1437	791	1353	914	544	535
Spd_RF2_80_133_1563	1668	1406	1914	1475	722	725
Spd_RF2_80_147_1571	1885	922	1677	995	550	474
Spd_RF2_80_147_1579	1326	930	1413	853	489	477
Spd_RF2_80_147_1587	1486	983	1648	976	657	602
Spd_RF2_80_147_1595	1559	1047	1515	985	555	535
Spd_RF2_80_147_1603	1626	1293	1664	1102	685	554

Tabla 9. Coste total obtenido en redes con 80 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_100_114_1811	2337	2262	2311	2211	1982	1970
Spd_RF2_100_114_1819	2384	2195	2277	1979	1892	1891
Spd_RF2_100_114_1827	2098	2020	2011	2148	1724	1721
Spd_RF2_100_114_1835	2092	2014	1905	1722	1653	1635
Spd_RF2_100_114_1843	2182	1879	1979	1987	1761	1734
Spd_RF2_100_129_1851	2382	2018	2360	2042	1597	1533
Spd_RF2_100_129_1859	2032	2100	2278	1982	1444	1396
Spd_RF2_100_129_1867	2060	2008	2201	1856	1489	1489
Spd_RF2_100_129_1875	2020	1831	2004	1772	1385	1403
Spd_RF2_100_129_1883	1948	1657	1777	1686	1205	1158
Spd_RF2_100_144_1891	2033	1783	1936	1660	1103	1099
Spd_RF2_100_144_1899	2084	1601	2132	1626	1118	1114
Spd_RF2_100_144_1907	1976	1929	2129	1786	1379	1376
Spd_RF2_100_144_1915	1665	1413	1921	1343	927	914
Spd_RF2_100_144_1923	1773	1686	2040	1670	1284	1234
Spd_RF2_100_159_1931	2175	1686	2312	1588	916	903
Spd_RF2_100_159_1939	1851	1351	2084	1218	982	907
Spd_RF2_100_159_1947	2155	1309	2016	1480	834	863
Spd_RF2_100_159_1955	1948	1487	2474	1500	913	850
Spd_RF2_100_159_1963	1924	1431	1857	1438	994	998
Spd_RF2_100_174_1971	1844	1235	2117	1628	865	827
Spd_RF2_100_174_1979	2232	1552	1964	1513	824	823
Spd_RF2_100_174_1987	1975	1153	2203	1525	834	820
Spd_RF2_100_174_1995	1818	1658	2180	1614	889	859
Spd_RF2_100_174_2003	1955	1273	2002	1394	684	670

Tabla 10. Coste total obtenido en redes con 100 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_120_136_2211	2664	2395	2442	2378	2299	2283
Spd_RF2_120_136_2219	2658	2495	2460	2396	2208	2169
Spd_RF2_120_136_2227	2513	2564	2830	2665	2190	2184
Spd_RF2_120_136_2235	2573	2441	2497	2346	2081	2091
Spd_RF2_120_136_2243	2601	2707	2746	2593	2220	2215
Spd_RF2_120_152_2251	2418	1944	2328	1941	1489	1453
Spd_RF2_120_152_2259	2411	2169	2528	1989	1765	1815
Spd_RF2_120_152_2267	2538	2177	2559	2244	1821	1809
Spd_RF2_120_152_2275	2269	2025	2407	2120	1748	1688
Spd_RF2_120_152_2283	2487	2275	2658	2165	2045	2050
Spd_RF2_120_169_2291	2903	2674	2904	2336	1704	1652
Spd_RF2_120_169_2299	2290	1836	2665	2090	1631	1591
Spd_RF2_120_169_2307	2251	1777	2039	1676	1328	1274
Spd_RF2_120_169_2315	2778	2479	2964	2196	1720	1673
Spd_RF2_120_169_2323	2412	1980	2559	1940	1549	1526
Spd_RF2_120_185_2331	2381	1692	2407	1685	1147	1089
Spd_RF2_120_185_2339	2554	1892	2571	1830	1488	1407
Spd_RF2_120_185_2347	2304	1625	2463	1823	1216	1196
Spd_RF2_120_185_2355	2434	1841	2317	1807	1198	1143
Spd_RF2_120_185_2363	2341	1725	2306	1795	1271	1175
Spd_RF2_120_202_2371	2292	1657	2121	1497	986	952
Spd_RF2_120_202_2379	2269	1876	2597	1947	1362	1338
Spd_RF2_120_202_2387	2083	1720	2438	1767	1029	1036
Spd_RF2_120_202_2395	2248	1432	2299	1620	859	830
Spd_RF2_120_202_2403	2200	1889	2571	1970	1111	1059

Tabla 11. Coste total obtenido en redes con 120 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_140_157_2611	2886	2886	2785	2836	2586	2583
Spd_RF2_140_157_2619	2887	2605	2619	2615	2366	2358
Spd_RF2_140_157_2627	3228	3041	3221	2948	2726	2705
Spd_RF2_140_157_2635	3397	3148	3207	3096	2828	2802
Spd_RF2_140_157_2643	2849	2834	2818	2818	2395	2391
Spd_RF2_140_175_2651	2968	2671	2714	2586	2182	2113
Spd_RF2_140_175_2659	3259	2610	3054	2604	2231	2223
Spd_RF2_140_175_2667	3487	2783	3129	2665	2256	2165
Spd_RF2_140_175_2675	3171	2732	3084	2752	2489	2441
Spd_RF2_140_175_2683	2743	2430	3086	2549	2074	2060
Spd_RF2_140_193_2691	2932	2534	2741	2393	1665	1630
Spd_RF2_140_193_2699	2832	2170	2627	2128	1567	1518
Spd_RF2_140_193_2707	2719	2365	2842	2314	1725	1705
Spd_RF2_140_193_2715	2748	2343	3148	2383	1848	1798
Spd_RF2_140_193_2723	2870	2307	2591	2148	1680	1602
Spd_RF2_140_211_2731	2960	2653	3145	2506	1619	1555
Spd_RF2_140_211_2739	2985	2493	3076	2313	1544	1527
Spd_RF2_140_211_2747	2673	2336	2758	2248	1498	1491
Spd_RF2_140_211_2755	2456	2237	2880	2117	1423	1344
Spd_RF2_140_211_2763	2522	2244	3128	2004	1460	1447
Spd_RF2_140_229_2771	2240	1796	2462	1649	1248	1240
Spd_RF2_140_229_2779	2738	1703	2945	1949	1194	1107
Spd_RF2_140_229_2787	2860	1987	2694	1701	1288	1206
Spd_RF2_140_229_2795	2671	1820	2845	2111	1285	1223
Spd_RF2_140_229_2803	2837	2102	3413	2239	1378	1342

Tabla 12. Coste total obtenido en redes con 140 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_160_179_3011	3517	3082	3117	3084	2849	2819
Spd_RF2_160_179_3019	3851	3698	3704	3590	3279	3237
Spd_RF2_160_179_3027	3825	3787	3612	3664	3309	3281
Spd_RF2_160_179_3035	3452	3189	3491	3294	3026	2995
Spd_RF2_160_179_3043	3106	3013	3062	2873	2829	2777
Spd_RF2_160_198_3051	3708	3174	3534	3366	2914	2868
Spd_RF2_160_198_3059	3279	3052	3205	2917	2338	2306
Spd_RF2_160_198_3067	3648	3237	3476	3122	2638	2602
Spd_RF2_160_198_3075	3740	2961	3056	2823	2435	2434
Spd_RF2_160_198_3083	3307	3074	3361	2912	2680	2670
Spd_RF2_160_218_3091	3534	2835	3346	2649	2082	2031
Spd_RF2_160_218_3099	3534	2872	3543	2834	2598	2550
Spd_RF2_160_218_3107	3499	2623	3084	2491	2090	1988
Spd_RF2_160_218_3115	3191	2919	3125	2847	2357	2320
Spd_RF2_160_218_3123	3230	3111	3425	2699	2262	2196
Spd_RF2_160_237_3131	3259	2805	3696	2701	1920	1848
Spd_RF2_160_237_3139	3101	2529	3359	2626	1954	1868
Spd_RF2_160_237_3147	3351	2758	3423	2584	1766	1797
Spd_RF2_160_237_3155	2854	2407	3305	2303	1767	1750
Spd_RF2_160_237_3163	3124	2384	3424	2396	1812	1777
Spd_RF2_160_257_3171	3318	2725	3239	2513	1932	1808
Spd_RF2_160_257_3179	3099	2375	3440	2406	1523	1490
Spd_RF2_160_257_3187	3137	2484	3158	2554	1552	1496
Spd_RF2_160_257_3195	2940	2224	2734	1925	1470	1454
Spd_RF2_160_257_3203	3164	2332	3457	2434	1748	1677

Tabla 13. Coste total obtenido en redes con 160 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_180_200_3411	4446	4265	4350	4125	3969	3903
Spd_RF2_180_200_3419	4091	3801	3960	3784	3583	3544
Spd_RF2_180_200_3427	4042	3908	3776	3860	3496	3417
Spd_RF2_180_200_3435	4142	4056	3869	3810	3637	3645
Spd_RF2_180_200_3443	4161	3955	3814	3890	3736	3688
Spd_RF2_180_221_3451	3731	3485	3719	3254	2985	2865
Spd_RF2_180_221_3459	3565	3187	3780	3040	2712	2643
Spd_RF2_180_221_3467	4237	3677	3945	3648	3178	3127
Spd_RF2_180_221_3475	3680	3755	3689	3281	2770	2694
Spd_RF2_180_221_3483	3786	3265	3756	3397	2902	2846
Spd_RF2_180_242_3491	4106	3240	3246	2677	2271	2181
Spd_RF2_180_242_3499	3805	3372	4331	3588	2492	2453
Spd_RF2_180_242_3507	3847	2999	3508	2746	2305	2176
Spd_RF2_180_242_3515	3622	2841	3962	2786	2243	2207
Spd_RF2_180_242_3523	3536	2959	3853	2781	2400	2354
Spd_RF2_180_263_3531	3649	2928	3812	2768	2159	2080
Spd_RF2_180_263_3539	3444	3010	3949	2965	2161	2111
Spd_RF2_180_263_3547	3520	2704	3764	2845	2127	2045
Spd_RF2_180_263_3555	3221	2815	3917	2641	1951	1911
Spd_RF2_180_263_3563	3744	3398	3638	3056	2249	2185
Spd_RF2_180_284_3571	3299	2612	3569	2184	1683	1465
Spd_RF2_180_284_3579	3585	2426	3440	2586	1774	1733
Spd_RF2_180_284_3587	3388	2715	3404	2541	1571	1556
Spd_RF2_180_284_3595	3447	2781	3644	2589	2174	2152
Spd_RF2_180_284_3603	3845	2982	3859	3053	2129	2040

Tabla 14. Coste total obtenido en redes con 180 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_200_222_3811	4258	4240	4613	4046	3947	3942
Spd_RF2_200_222_3819	4535	4371	4429	4361	3979	3949
Spd_RF2_200_222_3827	4174	4110	4459	4032	3807	3763
Spd_RF2_200_222_3835	4377	4073	4129	4144	3680	3609
Spd_RF2_200_222_3843	4228	4208	4265	4014	3888	3882
Spd_RF2_200_244_3851	4957	4147	4336	4077	3808	3792
Spd_RF2_200_244_3859	4308	4029	4373	3957	3282	3260
Spd_RF2_200_244_3867	4520	3823	4081	3486	3228	3192
Spd_RF2_200_244_3875	4266	3807	4063	3835	3177	3211
Spd_RF2_200_244_3883	4253	3866	4077	3715	3292	3214
Spd_RF2_200_267_3891	4154	3489	4061	3371	2837	2743
Spd_RF2_200_267_3899	3901	3268	3885	3284	2585	2543
Spd_RF2_200_267_3907	3932	3404	4363	3359	2579	2563
Spd_RF2_200_267_3915	4133	3576	4303	3649	2847	2714
Spd_RF2_200_267_3923	4311	3905	4427	3766	3060	2987
Spd_RF2_200_289_3931	4048	3365	4079	3230	2424	2400
Spd_RF2_200_289_3939	4164	3401	3949	3355	2659	2597
Spd_RF2_200_289_3947	4268	3367	4283	3159	2474	2237
Spd_RF2_200_289_3955	4003	3617	4193	3099	2769	2665
Spd_RF2_200_289_3963	4136	3154	3879	2965	2425	2341
Spd_RF2_200_312_3971	3967	2711	3645	2763	1930	1853
Spd_RF2_200_312_3979	4059	2917	4056	2755	1781	1707
Spd_RF2_200_312_3987	4248	3035	4542	3363	2304	2196
Spd_RF2_200_312_3995	4125	3255	3850	2918	1956	1914
Spd_RF2_200_312_4003	3567	2689	4081	2814	2142	2112

Tabla 15. Coste total obtenido en redes con 200 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_250_273_4011	5876	5348	5433	5259	5111	5091
Spd_RF2_250_273_4019	5712	5456	5544	5314	4931	4916
Spd_RF2_250_273_4027	6071	5565	5651	5590	5279	5199
Spd_RF2_250_273_4035	5570	5105	5099	4955	4833	4787
Spd_RF2_250_273_4043	5444	5226	5232	5200	4883	4831
Spd_RF2_250_297_4051	5533	5160	5256	5042	4517	4341
Spd_RF2_250_297_4059	5417	5229	5439	4980	4469	4423
Spd_RF2_250_297_4067	5029	4507	4689	4376	4069	4013
Spd_RF2_250_297_4075	5725	5023	5425	4868	4619	4566
Spd_RF2_250_297_4083	5479	5142	5499	4993	4424	4391
Spd_RF2_250_321_4091	5316	4620	5295	4707	3770	3703
Spd_RF2_250_321_4099	5248	4534	5306	4244	3583	3522
Spd_RF2_250_321_4107	5614	4966	5113	4832	4038	4006
Spd_RF2_250_321_4115	5044	4457	5536	4400	3997	3892
Spd_RF2_250_321_4123	5326	4781	5606	4964	4128	4071
Spd_RF2_250_345_4131	4894	3905	4858	3799	3145	3018
Spd_RF2_250_345_4139	5163	4302	5622	4417	3627	3508
Spd_RF2_250_345_4147	4999	4341	5412	4165	3405	3282
Spd_RF2_250_345_4155	5168	4229	4951	4004	3052	2997
Spd_RF2_250_345_4163	5293	4434	5244	4480	3582	3422
Spd_RF2_250_369_4171	5059	3901	5603	3894	2916	2748
Spd_RF2_250_369_4179	4795	4256	5339	3890	2522	2448
Spd_RF2_250_369_4187	4944	3777	4662	3804	2753	2672
Spd_RF2_250_369_4195	4316	3607	5202	3603	2656	2511
Spd_RF2_250_369_4203	4983	4104	5373	3992	3029	2947

Tabla 16. Coste total obtenido en redes con 250 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_300_326_4211	6813	6463	6434	6338	5975	5966
Spd_RF2_300_326_4219	6551	6399	6534	6387	6037	5961
Spd_RF2_300_326_4227	6657	6352	6429	6161	5856	5827
Spd_RF2_300_326_4235	7109	6764	7032	6834	6422	6320
Spd_RF2_300_326_4243	6497	6284	6461	6050	5774	5706
Spd_RF2_300_353_4251	7060	6198	6501	5985	5706	5610
Spd_RF2_300_353_4259	6266	5467	5738	5258	5150	5064
Spd_RF2_300_353_4267	6990	6264	6704	6279	5443	5347
Spd_RF2_300_353_4275	6549	5874	6303	5785	5248	5094
Spd_RF2_300_353_4283	6699	5935	6224	6001	5468	5316
Spd_RF2_300_380_4291	6911	5876	6953	5818	5387	5372
Spd_RF2_300_380_4299	6591	5923	6620	5746	4898	4892
Spd_RF2_300_380_4307	5680	5276	6060	5086	4151	4120
Spd_RF2_300_380_4315	6321	5173	6017	5172	4504	4364
Spd_RF2_300_380_4323	6138	5776	6681	5513	4515	4350
Spd_RF2_300_407_4331	6518	6033	6772	5904	4631	4456
Spd_RF2_300_407_4339	6545	5382	6729	5400	4331	4097
Spd_RF2_300_407_4347	6135	5255	6133	5265	4256	4127
Spd_RF2_300_407_4355	6564	5227	6569	5586	4293	4153
Spd_RF2_300_407_4363	5690	5183	6018	4933	4049	3962
Spd_RF2_300_434_4371	6225	4968	6838	5126	3752	3665
Spd_RF2_300_434_4379	6388	5114	5931	4936	3686	3523
Spd_RF2_300_434_4387	6399	5194	7080	5072	3908	3859
Spd_RF2_300_434_4395	5913	5266	6868	5167	4024	3908
Spd_RF2_300_434_4403	5859	4783	6265	4582	3704	3684

Tabla 17. Coste total obtenido en redes con 300 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_350_378_4411	7793	7389	7214	7130	6818	6779
Spd_RF2_350_378_4419	7985	7512	7523	7501	7071	7005
Spd_RF2_350_378_4427	7936	8202	8075	7821	7434	7419
Spd_RF2_350_378_4435	7626	7481	7858	7303	7155	7135
Spd_RF2_350_378_4443	7953	7625	7647	7524	7034	6983
Spd_RF2_350_406_4451	8122	7513	7893	7484	6752	6719
Spd_RF2_350_406_4459	7655	7253	7381	6874	6367	6313
Spd_RF2_350_406_4467	8211	7800	7891	7330	6815	6683
Spd_RF2_350_406_4475	7935	7391	7339	6907	6466	6267
Spd_RF2_350_406_4483	7484	6688	7220	6821	6299	6197
Spd_RF2_350_435_4491	7772	6824	7115	6456	5993	5967
Spd_RF2_350_435_4499	6518	5969	7180	5959	5248	5155
Spd_RF2_350_435_4507	7174	6426	7034	6174	5648	5457
Spd_RF2_350_435_4515	7410	6812	7504	6528	5834	5663
Spd_RF2_350_435_4523	7556	6645	6979	6492	5635	5488
Spd_RF2_350_463_4531	7101	6422	7388	5889	4876	4773
Spd_RF2_350_463_4539	7386	6185	7382	5945	4953	4874
Spd_RF2_350_463_4547	7508	6240	7178	5911	4903	4849
Spd_RF2_350_463_4555	7096	5745	6973	5754	4934	4821
Spd_RF2_350_463_4563	8101	6959	7238	6400	5608	5491
Spd_RF2_350_492_4571	7630	5711	6890	5676	4709	4532
Spd_RF2_350_492_4579	7798	6071	7883	6189	5094	4949
Spd_RF2_350_492_4587	7463	6105	8038	6069	4469	4339
Spd_RF2_350_492_4595	6877	6079	7525	6023	4400	4267
Spd_RF2_350_492_4603	7076	5835	7593	5674	4635	4499

Tabla 18. Coste total obtenido en redes con 350 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_400_429_4611	9010	8874	9036	8954	8522	8565
Spd_RF2_400_429_4619	9511	9282	8869	8663	8757	8710
Spd_RF2_400_429_4627	9164	8987	9051	8982	8661	8610
Spd_RF2_400_429_4635	9220	8687	8672	8435	8285	8243
Spd_RF2_400_429_4643	9165	8736	8874	8843	8340	8203
Spd_RF2_400_459_4651	8910	8242	9029	8294	7751	7607
Spd_RF2_400_459_4659	8739	8235	8340	7706	7231	7094
Spd_RF2_400_459_4667	9029	8260	8803	8171	7766	7660
Spd_RF2_400_459_4675	8963	8633	9134	8538	7665	7540
Spd_RF2_400_459_4683	9237	8465	8627	8284	7913	7863
Spd_RF2_400_489_4691	8956	8276	8181	7788	6921	6791
Spd_RF2_400_489_4699	8338	7396	7973	7009	6463	6373
Spd_RF2_400_489_4707	8636	7749	8261	7850	7044	6938
Spd_RF2_400_489_4715	8535	7809	8182	7444	6648	6522
Spd_RF2_400_489_4723	8880	8042	8752	7632	6787	6669
Spd_RF2_400_519_4731	8492	7382	8186	7263	6290	6110
Spd_RF2_400_519_4739	8438	7097	8213	6992	6120	5922
Spd_RF2_400_519_4747	8470	7717	8358	7251	6364	6175
Spd_RF2_400_519_4755	7986	7028	8195	6829	5761	5693
Spd_RF2_400_519_4763	7848	7165	8563	7044	6182	6052
Spd_RF2_400_549_4771	8915	6950	8383	6913	5843	5621
Spd_RF2_400_549_4779	8158	7180	9089	7442	5967	5708
Spd_RF2_400_549_4787	8500	6938	7822	6759	5581	5358
Spd_RF2_400_549_4795	8153	7033	8182	6423	5464	5382
Spd_RF2_400_549_4803	8719	7391	9001	7127	6152	6037

Tabla 19. Coste total obtenido en redes con 400 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_450_482_4811	10856	10514	10232	10229	10007	9916
Spd_RF2_450_482_4819	10849	10429	10500	10317	10065	9968
Spd_RF2_450_482_4827	10437	10155	10043	9701	9640	9531
Spd_RF2_450_482_4835	9387	9361	9137	9122	8796	8664
Spd_RF2_450_482_4843	10204	10118	10042	9833	9532	9390
Spd_RF2_450_515_4851	9968	9742	10084	9603	8918	8777
Spd_RF2_450_515_4859	10578	10010	10592	9913	9270	9147
Spd_RF2_450_515_4867	9630	9211	9617	8935	8400	8292
Spd_RF2_450_515_4875	9948	9543	10071	9487	8663	8533
Spd_RF2_450_515_4883	9195	8876	9330	8562	8083	8065
Spd_RF2_450_548_4891	9949	8695	9750	8746	7923	7764
Spd_RF2_450_548_4899	9844	9109	9638	8775	7973	7887
Spd_RF2_450_548_4907	10023	8698	8912	8370	7517	7511
Spd_RF2_450_548_4915	9368	8165	8870	8083	7410	7116
Spd_RF2_450_548_4923	9904	9160	9337	8800	7747	7580
Spd_RF2_450_581_4931	9685	8598	10027	8787	7414	7210
Spd_RF2_450_581_4939	9003	8115	9022	7981	6763	6634
Spd_RF2_450_581_4947	9042	7730	9532	7778	6715	6663
Spd_RF2_450_581_4955	10126	8342	9115	8224	7055	6914
Spd_RF2_450_581_4963	9892	8613	9802	8302	7183	7072
Spd_RF2_450_614_4971	9763	7618	9521	7800	6382	6137
Spd_RF2_450_614_4979	9191	7926	9135	7478	6364	6267
Spd_RF2_450_614_4987	9607	7552	9407	7467	6305	5897
Spd_RF2_450_614_4995	8971	7282	8869	7203	6284	6058
Spd_RF2_450_614_5003	9403	8050	9364	7695	6472	6369

Tabla 20. Coste total obtenido en redes con 450 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_500_534_5011	11745	11282	11372	11313	10982	10908
Spd_RF2_500_534_5019	11730	11230	11301	11137	10786	10716
Spd_RF2_500_534_5027	11716	11667	11992	11557	11141	11107
Spd_RF2_500_534_5035	11738	11619	11651	11436	10984	10953
Spd_RF2_500_534_5043	11962	11226	11432	11107	10884	10730
Spd_RF2_500_568_5051	11622	10656	11111	10280	9692	9422
Spd_RF2_500_568_5059	11028	10268	10370	9963	9490	9299
Spd_RF2_500_568_5067	11253	10037	10391	10162	9555	9477
Spd_RF2_500_568_5075	10855	10218	10511	10105	9393	9291
Spd_RF2_500_568_5083	10801	10231	10347	9995	9509	9430
Spd_RF2_500_603_5091	11010	9724	10618	9454	8989	8811
Spd_RF2_500_603_5099	10433	9905	10818	9621	8955	8774
Spd_RF2_500_603_5107	10784	9924	10746	9460	8732	8479
Spd_RF2_500_603_5115	11252	10523	11077	10228	9254	9025
Spd_RF2_500_603_5123	11308	10041	10588	9635	9015	8722
Spd_RF2_500_637_5131	10222	9488	10710	9227	8177	8025
Spd_RF2_500_637_5139	10909	9573	10063	8989	8447	8248
Spd_RF2_500_637_5147	10343	9493	10334	8929	7855	7757
Spd_RF2_500_637_5155	10005	8803	9420	8554	7445	7188
Spd_RF2_500_637_5163	10015	8949	10058	8401	7350	7230
Spd_RF2_500_672_5171	11450	8276	10311	8933	7657	7487
Spd_RF2_500_672_5179	9961	8700	10596	8548	7101	6896
Spd_RF2_500_672_5187	9389	8039	10060	8308	6571	6451
Spd_RF2_500_672_5195	10801	9570	10978	8952	8055	7808
Spd_RF2_500_672_5203	9652	8214	9507	7697	6718	6539

Tabla 21. Coste total obtenido en redes con 500 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_600_637_5211	14002	13919	14520	14152	13321	13271
Spd_RF2_600_637_5219	13958	13610	14340	13526	12918	12824
Spd_RF2_600_637_5227	13825	13300	13839	13573	12824	12757
Spd_RF2_600_637_5235	14319	13999	14517	13819	13241	13252
Spd_RF2_600_637_5243	14034	13867	14025	13967	13180	13050
Spd_RF2_600_674_5251	14362	14144	14640	14022	12916	12814
Spd_RF2_600_674_5259	14225	13619	14774	13755	12538	12456
Spd_RF2_600_674_5267	14775	14005	14999	14025	13247	13236
Spd_RF2_600_674_5275	13098	12497	13013	12576	11780	11695
Spd_RF2_600_674_5283	14376	14062	14907	14060	12970	12877
Spd_RF2_600_712_5291	13348	12489	14477	12894	11428	11284
Spd_RF2_600_712_5299	14509	13809	14916	13951	12320	12206
Spd_RF2_600_712_5307	14901	13534	15613	13871	12452	12163
Spd_RF2_600_712_5315	14491	13407	14056	13303	12128	12058
Spd_RF2_600_712_5323	13937	13393	14608	13221	12246	12167
Spd_RF2_600_749_5331	13811	12700	14390	12725	11201	11113
Spd_RF2_600_749_5339	12997	12125	13725	12629	10786	10651
Spd_RF2_600_749_5347	13744	12195	14317	12637	11353	11226
Spd_RF2_600_749_5355	13543	12335	14431	12804	11248	11049
Spd_RF2_600_749_5363	14070	12881	14795	13337	11581	11504
Spd_RF2_600_787_5371	13579	11994	14001	12123	10540	10440
Spd_RF2_600_787_5379	13449	12208	14322	12422	10467	10254
Spd_RF2_600_787_5387	12362	11410	13490	11746	10184	10020
Spd_RF2_600_787_5395	13659	12446	14311	12615	10689	10554
Spd_RF2_600_787_5403	13209	11998	14146	12506	10829	10699

Tabla 22. Coste total obtenido en redes con 600 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_700_740_5411	16766	16388	17030	16473	15718	15593
Spd_RF2_700_740_5419	16248	15996	16448	15913	15186	15277
Spd_RF2_700_740_5427	16358	16268	16698	16344	15529	15492
Spd_RF2_700_740_5435	16651	16377	16769	16493	15617	15474
Spd_RF2_700_740_5443	17279	16549	16955	16707	16086	16034
Spd_RF2_700_780_5451	17157	16671	17476	16613	15446	15304
Spd_RF2_700_780_5459	15650	14923	16071	14890	13974	13884
Spd_RF2_700_780_5467	16272	15412	16205	15662	14665	14498
Spd_RF2_700_780_5475	16341	15711	16672	15857	14644	14583
Spd_RF2_700_780_5483	16244	16313	17335	16014	14926	14867
Spd_RF2_700_821_5491	15131	14178	15681	14249	13091	12986
Spd_RF2_700_821_5499	16196	14514	16288	15084	13652	13466
Spd_RF2_700_821_5507	15950	14829	16353	14879	13832	13661
Spd_RF2_700_821_5515	15758	14366	16206	14602	13502	13243
Spd_RF2_700_821_5523	15884	14944	16505	15401	13510	13305
Spd_RF2_700_861_5531	15864	15066	16483	15356	13034	12767
Spd_RF2_700_861_5539	16294	15271	16422	15035	13447	13276
Spd_RF2_700_861_5547	15677	14756	16542	14819	13141	12926
Spd_RF2_700_861_5555	15925	14560	16786	15082	13288	13032
Spd_RF2_700_861_5563	15412	13988	16169	14460	12657	12506
Spd_RF2_700_902_5571	17010	15475	17052	15107	13781	13609
Spd_RF2_700_902_5579	15298	14319	16868	14558	12973	12713
Spd_RF2_700_902_5587	15691	14387	16120	14361	12730	12606
Spd_RF2_700_902_5595	15613	14505	16960	14367	12946	12765
Spd_RF2_700_902_5603	16234	14817	16909	14575	13019	12843

Tabla 23. Coste total obtenido en redes con 700 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_800_843_5611	19136	18788	19266	18836	18164	18102
Spd_RF2_800_843_5619	18754	17821	18088	17737	17269	17152
Spd_RF2_800_843_5627	19157	18846	19647	18702	18039	17887
Spd_RF2_800_843_5635	19521	19344	19484	19080	18489	18396
Spd_RF2_800_843_5643	18552	18143	18519	18315	17402	17370
Spd_RF2_800_886_5651	17946	17472	18868	17521	16594	16500
Spd_RF2_800_886_5659	17854	17042	17724	17149	15714	15563
Spd_RF2_800_886_5667	18786	18507	19247	17692	17006	16951
Spd_RF2_800_886_5675	18670	18388	19248	18384	17111	16947
Spd_RF2_800_886_5683	19149	17724	18984	18092	16842	16681
Spd_RF2_800_930_5691	18550	17593	19139	18200	16206	16034
Spd_RF2_800_930_5699	18671	17682	19752	17975	16332	15974
Spd_RF2_800_930_5707	18479	17549	18859	17323	15882	15746
Spd_RF2_800_930_5715	17776	16968	18614	16912	15602	15402
Spd_RF2_800_930_5723	18250	17682	18129	17453	15985	15715
Spd_RF2_800_973_5731	18648	17463	20001	17909	16123	16069
Spd_RF2_800_973_5739	18237	16904	18618	17258	15161	14991
Spd_RF2_800_973_5747	17742	16996	18913	17199	15061	14739
Spd_RF2_800_973_5755	18847	17448	19336	17867	15683	15576
Spd_RF2_800_973_5763	18523	17194	18495	17106	15153	15017
Spd_RF2_800_1017_5771	17357	15982	18056	15821	14192	13944
Spd_RF2_800_1017_5779	18631	17105	19514	17193	15316	15202
Spd_RF2_800_1017_5787	17409	15946	17919	16211	13912	13753
Spd_RF2_800_1017_5795	19126	17634	19378	17440	15195	15165
Spd_RF2_800_1017_5803	18263	17019	19012	17280	14885	14609

Tabla 24. Coste total obtenido en redes con 800 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_900_944_5811	21408	20759	21385	20981	20138	20021
Spd_RF2_900_944_5819	21091	21001	21243	20784	20250	20114
Spd_RF2_900_944_5827	22143	21552	21908	21426	20579	20435
Spd_RF2_900_944_5835	21873	21417	21870	21619	20511	20384
Spd_RF2_900_944_5843	21332	20832	20995	20876	19872	19737
Spd_RF2_900_989_5851	21965	21111	22524	21138	20011	19741
Spd_RF2_900_989_5859	21587	21023	21949	21200	19592	19357
Spd_RF2_900_989_5867	21263	20543	21570	20273	19382	19195
Spd_RF2_900_989_5875	20623	19993	21537	20042	18953	18857
Spd_RF2_900_989_5883	20790	19919	21412	20146	18467	18328
Spd_RF2_900_1034_5891	20183	19162	21039	19576	17953	17855
Spd_RF2_900_1034_5899	21239	20105	21712	20160	18759	18606
Spd_RF2_900_1034_5907	19773	19380	21422	19753	17959	17745
Spd_RF2_900_1034_5915	20450	19227	20779	19519	18085	17964
Spd_RF2_900_1034_5923	22003	20670	22610	20786	19420	19237
Spd_RF2_900_1079_5931	19832	18929	20810	18818	16981	16721
Spd_RF2_900_1079_5939	20957	19933	21608	19531	17652	17279
Spd_RF2_900_1079_5947	20741	19592	21401	20229	17630	17344
Spd_RF2_900_1079_5955	20443	18974	20659	19458	17570	17391
Spd_RF2_900_1079_5963	20409	19005	20768	19332	17477	17260
Spd_RF2_900_1124_5971	20262	18244	21370	18178	16665	16506
Spd_RF2_900_1124_5979	21256	19510	21731	19702	17574	17320
Spd_RF2_900_1124_5987	20093	18526	21552	18415	16648	16316
Spd_RF2_900_1124_5995	20491	18666	21123	18880	17005	16794
Spd_RF2_900_1124_6003	19738	18567	21224	18882	16404	16292

Tabla 25. Coste total obtenido en redes con 900 nodos.

GRAFOS	CLÁSICO	DINÁMICO	HEU – N1	HEU - N2	HEU - N3	GENÉTICO
Spd_RF2_1000_1047_6011	24022	23648	24394	23784	23033	22957
Spd_RF2_1000_1047_6019	23761	23808	23854	23449	22624	22577
Spd_RF2_1000_1047_6027	24077	24058	24127	24046	22933	22761
Spd_RF2_1000_1047_6035	23331	22796	23474	22897	22215	22042
Spd_RF2_1000_1047_6043	24590	23894	24228	24180	23228	23042
Spd_RF2_1000_1095_6051	22845	22203	23323	22297	20836	20575
Spd_RF2_1000_1095_6059	23448	22361	23999	22403	21351	21296
Spd_RF2_1000_1095_6067	23642	22884	24083	22978	21160	21054
Spd_RF2_1000_1095_6075	23286	22929	23083	22515	21031	20684
Spd_RF2_1000_1095_6083	23735	22356	23626	22602	21467	21306
Spd_RF2_1000_1143_6091	24750	23659	25439	24024	22140	22113
Spd_RF2_1000_1143_6099	23381	22664	24877	22940	20853	20866
Spd_RF2_1000_1143_6107	22617	21614	23406	22169	20256	19981
Spd_RF2_1000_1143_6115	24058	22612	23561	22679	21051	20872
Spd_RF2_1000_1143_6123	22646	21057	22866	21651	19898	19818
Spd_RF2_1000_1191_6131	23016	22182	23661	22177	19782	19520
Spd_RF2_1000_1191_6139	23319	22472	24999	22345	20181	20030
Spd_RF2_1000_1191_6147	23317	22457	24378	22213	20125	19948
Spd_RF2_1000_1191_6155	23686	22415	24628	22442	20060	19899
Spd_RF2_1000_1191_6163	22426	21253	23600	21168	19386	19260
Spd_RF2_1000_1239_6171	22275	20418	23524	21008	18528	18333
Spd_RF2_1000_1239_6179	22170	20132	23000	20760	18396	18168
Spd_RF2_1000_1239_6187	23229	21890	23675	21222	19390	19168
Spd_RF2_1000_1239_6195	21521	20296	23133	20309	18544	18152
Spd_RF2_1000_1239_6203	22593	20334	23543	21302	18571	18365

Tabla 26. Coste total obtenido en redes con 1000 nodos.

6. BIBLIOGRAFÍA

[1] Cisco Systems. (2018). Cisco Visual Networking Index: Forecast and Trends, 2017–2022. Cisco White Papers.

[2] Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal, 36*(6), 1389–1401.

[3] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society, 7*(1), 48–50.

[4] Du, D. Z., & Pardalos, P. M. (Eds.). (1998). Spanning trees with degree constraints. In *Handbook of Combinatorial Optimization* (pp. 123–158). Springer.

[5] Martello, S., & Toth, P. (1990). On variations of the minimum spanning tree with vertex costs. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley.

[6] International Energy Agency. (2020). *Data Centre Energy Efficiency and Electrification*. IEA Reports.

[7] Cisco Systems. (2019). *Global Cloud Index 2018–2023*. Cisco White Papers.

[8] Baker, F. (2020). *Internet Almanac 2020: ISP and Carrier Profiles*. Internic.

[9] Wang, X., Lim, L., Zhu, S., & Wu, H. (2002). Designing ISP backbone networks: Optimizing router placement and link costs. *Computer Networks, 38*(5), 705–719.

[10] Li, Y., Chua, K., & Wong, A. (2016). Quality-of-Service routing in ISP networks: Algorithms and implementation. *IEEE/ACM Transactions on Networking, 24*(1), 185–198.

[11] Du, D. Z., & Pardalos, P. M. (2018). Vertex-weighted spanning tree problems: A survey. *International Journal of Metaheuristics, 8*(2), 145–182.

[12] Guha, S., & Munagala, K. (2002). Approximation algorithms for partial cover problems. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)* (pp. 767–775). ACM.

[13] Kreutz, D., Ramos, F., Veríssimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A comprehensive survey. *Proceedings of the IEEE, 103*(1), 14–76.

[14] Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[15] Li, X., Pan, D., & Wang, Z. (2008). Efficient heuristics for degree-constrained minimum spanning tree. *Computers & Operations Research, 35*(1), 38–54.

[16] Uhlig, S., et al. (2019). IA in ISP-scale routing: Challenges and directions. *IEEE Communications Surveys & Tutorials, 21*(1), 66–120.

[17] Dhamdhere, A., & Dovrolis, C. (2011). ISP traffic engineering: Existing practices and remaining challenges. *IEEE Communications Magazine, 49*(10), 86–92.

[18] Rosen, E., & Rekhter, Y. (2002). BGP/MPLS VPNs. *IEEE/ACM Transactions on Networking, 10*(5), 637–648.

[19] Feamster, N., et al. (1999). Practical network support for ISP traffic engineering. *ACM SIGCOMM Computer Communication Review, 29*(2), 39–52.

[20] Chaskar, H., et al. (2017). Control plane and data plane scalability in software-defined access networks. *IEEE Communications Magazine, 55*(11), 106–113.

[21] Zhang, D., Pecheux, C., & Frenot, F. (2017). Designing resilient ISP networks: Principles and practice. *Journal of Network and Systems Management, 25*(1), 75–94.

[22] Ferreira, A. L., Xiao, X., & de M. Bicça, R. F. (2011). A mixed-integer programming formulation for minimum spanning trees with node degree constraints. *Journal of Combinatorial Optimization, 21*(1), 39–58.

[23] Golden, B. L., & Wasil, E. A. (2013). *Combinatorial Optimization: Analysis and Algorithms*. Springer.

[24] Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

[25] Hooker, J. N. (1992). A Lagrangian relaxation approach for minimum spanning tree problems with degree constraints. *Operations Research Letters, 12*(1), 1–5.

[26] Du, D. Z., & Pardalos, P. M. (Eds.). (1998). Degree-constrained spanning trees. In *Handbook of Combinatorial Optimization* (pp. 123–158). Springer.

[27] Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness* (2nd ed.). W. H. Freeman.

[28] Martello, S., & Toth, P. (1990). Heuristics and local search for spanning tree problems. In *Knapsack Problems: Algorithms and Computer Implementations*. Wiley.

[29] Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

[30] Kung, S. Y. (2012). Genetic algorithms for network optimization. *Applications of Artificial Intelligence in Telecommunications*. Wiley.

[31] Beneke, J., & Gommans, L. (2011). Network design optimization using genetic algorithms. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 3415–3421). IEEE.

[32] Latifi, A., Arabnia, H. R., & Gunduz, K. (2012). A survey of network optimization: Routing, design, and resilience. *International Journal of Network Management, 22*(1), 1–27.

[33] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942–1948). IEEE.

[34] Del Corso, L., Patrignani, M., & Pellegrini, M. (2010). A swarm intelligence approach for multi-objective network design. *Expert Systems with Applications, 37*(12), 7669–7675.

[35] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

[36] Pióro, M., & Medhi, D. (2004). *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann.

[37] Ferreira, A. L., Xiao, X., & de M. Bicça, R. F. (2011). A mixed-integer programming formulation for minimum spanning trees with node degree constraints. *Journal of Combinatorial Optimization, 21*(1), 39–58.

[38] Church, R. L., & Scarf, R. H. (1958). Optimal location of distribution centers. *Naval Research Logistics Quarterly, 5*(1), 495–506.

[39] Beraldi, P. G., & Liò, P. (2011). A capacitated facility location problem for designing backbone networks. *European Journal of Operational Research, 212*(2), 248–257.

[40] Drezner, Z. (Ed.). (1995). *Facility Location: A Survey of Applications and Methods*. Springer.

[41] Du, D., & Hu, P. M. (2012). *Combinatorial Optimization: Theory and Algorithms*. Springer.

[42] Nickel, C., & Saldanha da Gama, T. (2007). Redundant facility location. In *Semi-infinite Programming* (pp. 123–158). Springer.

[43] Landete, M., Marín, A., & Sainz-Pardo, J. L. (2019). Locating switches. *Expert systems with applications, 136,* 338-352.

[44] Savelsbergh, M. (2002). Preprocessing and relaxing bounds in mixed-integer programming: A survey. *Discrete Applied Mathematics, 123*(1–3), 271–294.

[45] Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed.). Springer.

[46] Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming, 98*(1–3), 23–47.

[47] Douligeris, A., & Sidiropoulos, N. D. (1997). Facility location and demand assignment in backbone network design. *IEEE/ACM Transactions on Networking, 5*(3), 371–376.

[48] Holmberg, P., & Paterson, M. (1997). Location problems for telecommunications networks. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)* (pp. 1–8). ACM.

[49] Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

[50] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

[51] Zitzler, E., Laumanns, M., & Thiele, L. (2002). SPEA2: Improving the strength Pareto evolutionary algorithm. In *Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN 2001)* (pp. 95–100). Springer.

[52] RIPE NCC & CAIDA. (2020). *Internet Topology Data Kit* [Data set]. Extraído de: https://www.ripe.net/manage-ips-and-asns/db/

[53] McKenney, P. (2018). *Bridging and Switching Fundamentals*. Cisco Press.

[54] Feamster, N., & Rexford, J. (2012). Network innovation through virtualization: A survey. *IEEE Communications Surveys & Tutorials, 14*(4), 1349–1361.

[55] Sezer, S., et al. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine, 51*(7), 36–43.

[56] Van Rossum, G. (2016). Hierarchical network topology design: Core, distribution and access layers. *Journal of Network Architecture, 5*(1), 45–58.

[57] Greenberg, A., et al. (2011). VL2: A scalable and flexible data center network. *Communications of the ACM, 54*(3), 95–104.

[58] Lamarca, A., et al. (2017). High-speed switching technologies: ASICs and NPUs. *IEEE Micro, 37*(3), 40–50.

[59] Pierre, M. (2011). Evolution of Ethernet switching from 10 Mbps to 100 Gbps. *IEEE Communications Magazine, 49*(11), 134–141.

[60] Heller, B., et al. (2015). Eliminating network protocol-induced queuing delays using modern switch hardware. *Proceedings of ACM SIGCOMM* (pp. 38–51).

[61] McKeown, N., et al. (2008). OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69–74.

[62] Bosshart, P., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review, 44*(3), 87–95.

[63] Marín, A. (2015). Exact and heuristic solutions for the Minimum Number of Branch Vertices Spanning Tree Problem. *European Journal of Operational Research, 245*, 680-689.

[64] Carrabs, A., et al. (2013). Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications, 56*, 405–43